DESIGN AND ANALYSIS OF OBJECT REPLACEMENT POLICES ON DYNAMIC DATA ALLOCATION AND REPLICATION ALGORITHM WITH BUFFER CONSTRAINTS

GU XIN

(*B.Eng., Beijing University of Aeronautics and Astronautics, PRC*)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2003

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Assistant Processor Dr. Veeravali Bharadwaj. Without his insightful ideas, valuable suggestions and constant encouragement, I could not have accomplished my research work. I want to deliver my sincere esteem to his rigorous research style, which led me to a correct research attitude. I am also very grateful to my supervisor for his help during the time I were receiving medical treatments in the hospital. Without his continuous and persistent support my research would not have been possible.

My wholehearted thanks to my family for their endless encouragement and self-giving love throughout my life. Deepest thanks to all my friends and lab mates in Open Source Software Lab for their ideas and support. The friendship with them makes my study in NUS a pleasant journey.

My special thanks to the National University of Singapore for granting me Research Scholarship and the Open Source Software Lab for providing an efficient working environment and facilities.

Finally, I would like to thank all those who granted me directly and indirectly help during the course of my research study with their inputs and support.

Contents

List of Figures

List of Tables

Summary

In the theoretical computer research areas, there has been lots of works on distributed database system. As a part of research in the domain of distributed object management, the basic idea underlying this thesis is to design efficient data allocation algorithms to minimize the total servicing cost for an arbitrary request schedule which includes read requests and write requests. In all works so far, however, the available resources at the single site or processor are considered to be infinite. For example, the available local database buffer size to store the replicas of the object at a site is assumed to be plentiful. However, in practice, each processor has only a finite local database buffer capacity to hold the copies of the object. When the available buffer space in a site is not enough to store a new copy of an object, the decision has to be made by each processor, for example, to evict an object copy in use to give space to the new copy. Thus we are naturally faced with a problem of allocating and replicating the object with the consideration of local database buffer constraints. For a distributed database system where each processor has only finite-size local database, we analyze the allocation strategies with revised model of Dynamic Window Mechanism (DWM) algorithm jointly implemented with three different types of object replacement strategies. If optimal utilization of this distributed system is expected, it is suggested to apply an allocation and replication algorithm to obtain the possible minimum cost for servicing the read-write request schedule. For this goal, DWM is designed to dynamically alter the allocation scheme of the object such that the cumulative cost of all operations involved in servicing read and write

requests is minimized. Three different object replacement strategies work jointly with DWM to deal with the situation wherein processors' local database buffer size is limited. We will show the impact on the allocation and replication strategies due to the limited local database storage capacities. The performances of different algorithms are analyzed theoretically and experimentally. We consider the competitive performance of different algorithms and present algorithms with their competitive ratios. In a general sense, we consider the above mentioned scenario in a model where the object sizes are assumed to be equal. We also consider the situation in which the object sizes are different from each other. Thus we attack the problem in a more generalized scenario.

Chapter 1

Introduction

Distributed database system (DDBS) technology is one of the major developments in the database systems area. There are claims that in the near future centralized database management will be an "antique curiosity" and most organizations will move toward distributed database management [1]. The intense interest in this subject in the research community supports this claim. Distributed database management system (DBMS) thus play an increasingly important role and has attracted more and more research efforts since the last two decades. The design of a distributed database management system involves making decisions on the placement of *data (object)* and *programs* across the sites of a computer network. The distribution of application programs is not a significant problem, since we assume that a copy of the distributed DBMS software exists at each site where data is stored [2]. Therefore, extensive studies are concentrated on data(object) distribution problem, which is widely known as data(object) allocation and replication problem.

In a distributed database, an object is usually desirable to be replicated in the local database of multiple locations for performance, reliability and availability reasons [2, 9, 10]. The object is accessed, i.e. read or written, from multiple distributed processors. These reads and writes form a set of requests, which is usually serialized by some concurrency-control mechanism [11]

in order that each read request accesses the most recent version of the object(written by a most recent write request). Such replication helps performance since diverse and conflicting user requirements can be easily accommodated. For example, an object that is commonly read by one processor can be placed on that processor's local database. This increases the locality of reference. Furthermore, if one of the processors fails, a copy of the object is still available on another processor on the network. If we focus on the servicing cost of the set of read-write requests for a replicated object, the cost of servicing a read or a write depends on the *allocation scheme* of the object, which is a set of processors that store the most recent version of the object in their local database. This is because that if a processor in the allocation scheme of an object issues a read request for this object, the read will be serviced locally and reading an object locally is less costly than reading it from a remote location. On the other hand, the execution of a write request may cause trouble since it usually write to all or a majority of copies of the object. Hence, the decision regarding data allocation and replication is a trade-off which depends on the read-write pattern for each object. if the read-write patterns change dynamically, in unpredictable ways, a dynamic allocation scheme of an object is preferred since it changes as the read-write requests are serviced.

In addition, the data allocation and replication can be discussed in the larger context of dynamic allocation. For example, the rapid growth of internet and World Wide Web is moving us to a distributed, highly interconnected information system. In such systems, an object (a document, an image, a file, raw data, etc.) is accessed from multiple distributed locations. The allocation and replication of objects in such distributed system has crucial effects on the system performance. Thus, all kinds of dynamic allocation problem can also be performed by a large category, say Distributed Object Management (DOM) algorithms [11]. A DOM algorithm maps each request to a set of processors to execute the request and it determines the allocation scheme of the object upon the servicing of requests at any point in time.

1.1 Related Work

Performance and reliability are the two major purposes of data allocation and replication. Our work addresses the former. Traditional performance oriented works on data allocation consider the static fashion, namely establishing an allocation scheme that will optimize performance, but will remain fixed until manual reallocation is executed. It has been studied extensively in the literature [40]. This problem is also called file allocation problem and the 1981 survey paper by Dowdy and Foster [41], dealing with the file allocation problem, cites close to a hundred reference. In contrast, our work keeps the allocation scheme dynamic during servicing of the requests.

In the theoretical computer science community there has been work on online algorithms [6], particularly for paging [42], searching [42] and caching [43]. Upon the analysis of online algorithms, competitiveness and convergence are two criteria for evaluating online algorithms. A competitive algorithm may not converge to the optimal allocation scheme when the read-write pattern is fixed or stabilizes, but a convergent algorithm may unboundedly diverge from the optimum when the read-write pattern is irregular. A competitive online algorithm is more appropriate for chaotic read-write patterns in which the past access pattern does not provide any indication to the future read-write pattern. In contrast, a convergent online algorithm is more appropriate for regular read-write patterns. [6], [42] and [43] are some early works that addressed competitiveness analysis for online algorithms. The work in this thesis also uses competitive ratio to analyze the online algorithm.

The data allocation and replication algorithms developed in [13] are examples for convergence rather than competitiveness. Assume that the pattern of access to each object is generally regular. Then, the convergent algorithms will move to the optimal allocation scheme for the global read-write pattern. The model there as well as in [14] and [17] considers only communication and ignores the I/O cost and availability constraints. A different adaptive

data replication algorithm is developed in [12]. The algorithm, called ADR, is adaptive in the sense that it changes allocation scheme of the object as changes occur in the read-write pattern of the object. ADR is also a convergent algorithm. Both the algorithms in [13] and [12] depend on the communication network having a specific tree topology.

A competitive dynamic data allocation and replication algorithm is presented in [17]. But this algorithm ignores I/O cost and t-available constraint, which is a constraint that guarantees a minimum number of copies of the object in the system at any point in time.

Another important competitive data allocation algorithm, called DA, is proposed in [11]. In [11] a mathematical model that is suitable for stationary computing environment is introduced for evaluating the performance of data allocation and replication algorithms in distributed database. Authors studied dynamic allocation as an independent concept, unrestricted by the limitations of a particular system, protocol, or application, and also considered caching in a peer to peer rather than client-server environment. Competitive analysis was used to establish significant relationship that indicates the superiority of static allocation and dynamic allocation respectively.

Another research area that is relevant to my study in this thesis is caching management in various contexts, e.g. internet and the World Wide Web, to quote some [29, 30, 31, 32, 33]; database disk buffering, e.g. [35, 36]; web proxy of World Wide Web, e.g. [37, 34]; and Client/Server databases, e.g. [27, 28]. Related introduction and studies on concurrent control, which is an essential module to account request serializability, were presented in [23, 24, 25].

1.2 Issues To Be Studied and Main Contributions

In the research area of data allocation and replication, a data allocation and replication algorithm solves three fundamental questions: Which object should be replicated? How

many replicas of each object are created? Where should the replicas of an object be allocated? Depending on different answers, different data allocation strategies are devised. In all works so far in data allocation and replication, the available resources at the processing site of distributed database system are always considered to be plentiful. For instance, the available local database buffer size to store the replicas of each object is assumed to be infinite. However, in reality, local database capacity at a processor is of finite size. When a processor's local database buffer is full while an allocation and replication scheme informs this processor of the need to save a newly requested object in its local database, we are naturally confronted with a problem of how to deal with this newly requested object. Should it be saved or not? Where and how should it be saved? What kind of effects does it have on the running data allocation algorithm? In this thesis, we consider the above mentioned scenario in which each of the processors in the distributed database system has a local database of finite size.

In this thesis, we analyze the cost of servicing a set of read-write requests for a replicated object and propose a mathematical model. With this mathematical model, the cost of servicing a read or a write request depends on the allocation scheme of the object which is a set of processors that store the most updated replicas of the object in their local databases. By using this model, we design and analyze a dynamic data allocation algorithm that adapts to the changes in the request patterns. This dynamic algorithm uses a windowing mechanism, and hence, we refer to this scheme as *dynamic window mechanism*(DWM). The key idea of DWM algorithm is to divide the read requests into saving-read requests and non-saving-read requests based on whether this division is able to minimize the total cost of servicing all the requests in a request sub-schedule. As an added constraint, our data allocation algorithm is working under the situation wherein processors in distributed database have limited local database capacities, which is reflective of a real-life situation. When a processor's local database is full and DWM decides that a new object for this processor should be replicated in its local database, we propose three different strategies to tackle this problem. Strategy I is No Replacement

(NR). Both Strategy II, namely Least Recently Used (LRU), and Strategy III, namely Least Frequently Used (LFU), will pick up and evict some existing objects in processor's local database buffer to give space for the new objet copy, thus these two strategies are put into one category, namely DWM-Replacement, in our later theoretical analysis. The difference between Strategy II and Strategy III is that they choose different evicting object candidate which has least "bad" effects on the total servicing cost of all requests. With the implementation of three strategies, we actually propose three versions of DWM based algorithms to attack dynamic data allocation with buffer constraints, namely DWM-No Replacement (DWM-NR), DWM-Least Recently Used (DWM-LRU) and DWM-Least Frequently Used (DWM-LFU).

We use competitive analysis, which is a widely used analytical methodology to evaluate online computation and online algorithms, to analyze and compare the performance of different algorithms. By using this tool, we perform a worst-case comparison of an online algorithm to an optimal, ideal, offline algorithm. By establishing cost functions for different strategies, we not only obtain competitive ratios for different algorithms, but also show the superiorities of different algorithms according to their competitiveness. We also use experimental analysis to compare the performances of the proposed algorithms to each other. Rigorous simulation experiments are carried out to validate the theoretical findings.

In a more general sense, we consider the above mentioned scenario in which the object sizes are assumed to be equal. This is referred to as Model A (Homogenous Object Sizes) in our thesis. We also consider the situation where the object sizes are different from each other. Thus we attack the problem in a more generalized scenario. This situation is referred to as Model B (Heterogenous Object Sizes). For Model B, besides Strategy I (No Replacement), we design another two object replacement algorithms which are able to deal with the situation wherein objects are of different sizes. The newly developed algorithms are logical extensions of LRU and LFU. We denote them by Heterogeneous object sizes LRU(LRU^{het}) and Heterogenous object sizes LFU(LFU^{het}). Accordingly, the DWM based algorithms used in Model B are

DWM-No Replacement (DWM-NR), DWM Heterogeneous object sizes LRU (DWM-LRU$^{\text{het}}$) and DWM Heterogeneous object sizes LFU (DWM-LFU$^{\text{het}}$).

1.3 Organization of the Thesis

The rest of the thesis is organized as follows:

In Chapter 2, we present the system model for data allocation and replication, we formalize the cost function. In addition, some preliminary definitions and notations are introduced.

In Chapter 3, we investigate the data allocation problem with buffer constraints. We first present the dynamic online algorithm, DWM. We also describe the object replacement strategies according to different environment models.

In Chapter 4, using competitive analysis, we state and prove competitive ratio of different proposed strategies. Through cost comparison, we summarize the superiorities of different strategies.

In Chapter 5, performances of the different proposed algorithms are studied using the simulation experiments. The observations are provided and useful discussion is also given in this section.

In the last part, Section 6, we highlight the conclusion and give direction of our future work.

Chapter 2

System Modeling

In this chapter, we first introduce the concepts of distributed database system and distributed database management system. Then, we describe the system model with discussions of its components. In addition, definitions and notations that will be frequently used throughout the thesis are introduced.

2.1 Distributed Database Systems

In recent years research and practical applications in the area of distributed systems have developed rapidly, stimulated by the significant progress in two fields of computer science, namely computer networks and database systems. Some of the most advanced types of distributed systems are *Distributed database systems* (DDBS). Distributed database system technology is one of the major developments in the database systems area. We can define a distributed database as a collection of multiple, logically interrelated local databases distributed and interconnected by a computer network [2]. In both hardware and software aspects, a DDBS may span many platforms, operating systems, communication protocols and database architectures. A general DDBS is illustrated in Fig.2.1.

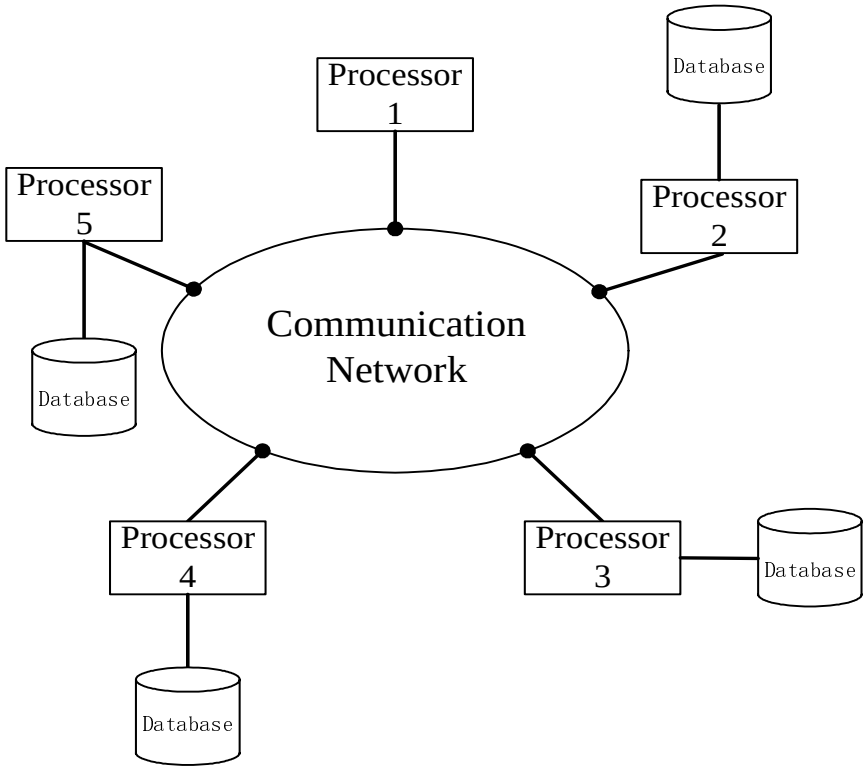


Figure 2.1: DDBS environment

A DDBS is managed by a Distributed Database Management System (DBMS). A DBMS is defined as the software system that permits the management of the DDBS and gives the users a transparent view of the distributed structure of the database [2]. New problem arises in distributed database systems, in comparison with centralized database systems, in terms of their management. One of the key problems is related to data allocation and replication. In a distributed environment, physical distribution of data is a significant issue in that it creates problems that are not encountered when the databases reside in the same computer. Data may be replicated in distributed sites for reliability and efficiency considerations. Consequently, it is responsible for a distributed database to have algorithms that analyze the request queries and convert them into a series of data manipulation operations. The problem is how to decide on a strategy for executing each request over the network in the most cost-effective way. The factor to be considered are the distribution of data, communication costs , and lack of sufficient locally available resources. The objective is to minimize the servicing cost

and improve the performance of executing the transaction subject to the above-mentioned constraints.

2.2 The Model

In this section, we first introduce our DDBS system and present the concept of request schedules and allocation schemes. In addition, definitions and notations that will be frequently used throughout the thesis are introduced. It is assumed that all the processors considered here have only finite size local database storage buffer. We also formulate the basic cost function on which our analysis of proposed algorithms is based on.

2.2.1 Request schedules and allocation schemes

In this thesis, our distributed database system consists of m processors, denoted by $p_1, p_2, ..., p_m$, which is interconnected by a message passing network to provide inter-processors communications. The local database is a set of objects stored in local database buffer of a processor. We assume that there is a Central Control Unit (CCU) in the system that knows the allocation scheme of every object and every processor knows whether or not an object is available in its local database buffer. Transactions operate on the object by issuing read and write requests. Requests arrive at the system concurrently and there exits a Concurrent Control Mechanism (CCM) to serialize them. A finite sequence of read-write requests of the object o, each of which is issued by a processor, is called a *request schedule*. For example, $\psi_o = r^2w^4r^1w^3r^2$ is a schedule for object o, in which the first request is a read request from processor 2, the second request is a write request from processor 4, etc. In practice, any pair of writes, or a read and a write, are totally ordered in a schedule; however, reads can execute concurrently. Our analysis using the model applies almost verbatim even if reads between two consecutive

writes are partially ordered.

In our system, we denote an *allocation scheme* as the set of processors that store the latest version of the object in their local database. The allocation scheme of an object is either dynamic or static, which means it either remains fixed or changes its composition as the read-write requests are serviced. The *initial allocation scheme* is given by a set of processors that have the object in their local database before the request schedule begins to be serviced. At any point of time the *allocation scheme at a request* q for object o is a set of processors that have the replica of object o in their local database right before q is executed, but after the immediately-preceding request for o is serviced. We denote the initial allocation scheme for object o as IA_o and let A_o be the allocation scheme of object o at a request at any point in time when the request schedule is executed. The initial allocation scheme IA_o consists of a set F of $(t-1)$ processors, and a processor p that is not in F. The processors of F are called the *main servers*, and p is called the *floating processor*. The number of processors in $F \bigcup \{p\}$ is t, which is referred to as *t-availability* constraint. Formally, *t-availability* constraint is defined as, for some integer t which is greater than 1, the allocation scheme at every request is of size which is at least t. In other words, t represents the minimum number of copies that must exist in the system. For simplicity, we assume that t is at least two. We shall also assume that t is smaller than the total number of processors in the network, otherwise, each new version of the object created by a write request must be propagated to all the processors of the network, and there is no need to address the problem with dynamic allocation algorithm.

When servicing the request schedule, each read request brings the object to main memory of processor which issued this request in order to service this request. If this processor does not have a copy of the object in its local database buffer, in case of dynamic allocation, the reading processor, s, may also store the object in the local database in order to service future reads at s locally. We denote a read request as a *saving read* which results in saving the object in the local database. A read request which does not result in saving the object in the local

database is denoted as a *non-saving read* request.

A write request in a schedule creates a new version of the object. Given a schedule, the latest version of the object at a request q is the version created by the most recent write request that proceeds q. The write request also sends the new version of object to all servers in F and each server outputs the object into its local database. If the processor s which issued a write request is a server in F, then s also sends a copy of the object to the floating processor in order to satisfy the *t-availability constraint*. Additionally, the write request results in the invalidation of the copies of the object at all other processors since these copies are obsolete. We summarize the effect of a write by considering an allocation scheme A immediately after a write request from a processor s. If s is in F, then $A = F \bigcup \{p\}$, and if s is not in F, then $A = F \bigcup \{s\}$. The following example illustrates the execution of a request schedule and the alteration of allocation scheme.

Example 2.1: Consider the request schedule $\psi_o = r^2 w^4 r^1 w^3 r^2$ given above and the initial allocation scheme $\{1, 2\}$ in which 1 is the main server set F and 2 is the floating processor. The allocation scheme at the first request r^2 is {1,2}; the allocation scheme at the second request w^4 is still {1,2} since processor 2 service the read request locally. The allocation scheme at the third request is {1,4}, and the processor 4 enters the allocation scheme after a write request (by processor 4) is serviced. The allocation scheme remains unchanged at the forth request w^3. Also, w^3 makes processor 3 enter the allocation scheme, thus after finishing the service of w^3, the allocation scheme at the last request r^2 is {1,3}.

2.2.2 Cost model

In this section, we present the cost model widely used in analysis of object allocation and replication in distributed database system. The performance matric in this thesis is the cumulative cost of all the service behaviors related to handling read and write requests which

arrive to the distributed database system. There are three types of costs associated with servicing the access requests. The first one is the I/O cost, i.e., the cost of inputting the object from the local database buffer to the processor's main memory or outputting the object from the processor's main memory to the local database. The I/O cost of servicing an access request is denoted by C_{io}. This means that the average I/O cost of a read or a write in the distributed system is C_{io}. The other two types of costs are related to the communication cost in the interconnected network, namely, the passing of control-messages and data-messages. An example of a control-message is a request message issued by a processor p to request another processor q to transfer a copy of an object which is not in p's local database to p. The data-message is merely a message in which the object is transmitted between the processors via networks. Different costs are associated with the two types of messages. We denote the communication cost of a control-message by C_c and the communication cost of a data-message by C_d.

In addition, there is another kind of cost which comes from object invalidation operation. This operation intends to invalidate the copies of an object o. Since the purpose of this invalidation operation is to inform the certain processor to invalidate the corresponding object, only control messages need to be passed. Thus, the cost of invalidation operation is equal to C_c. As the fact that the size of a control-message is much shorter than a data-message since a control-message consists of an object-id and operation types (read, write or invalidate) while a data-message also includes the copy of object besides the object-id and operation types, and the fact that the I/O operation is a local behavior which does not utilize any external resources, it is reasonable to assume $C_d \geq C_c \geq C_{io}$. It may be noted that a control message and an invalidation message are issued by the CCU unit. Additionally, observe that this definition assumes a homogeneous system in which the data-message between every pair of processors costs C_d, the control-message between every pair of processors costs C_c, and the I/O cost is identical at all processors.

In the existing literature [11, 17], without loss of generality, the I/O cost is normalized to be one unit ($C_{io} = 1$). This means that C_c is the ratio of the cost of transmitting a control message to the I/O cost of a read-write request. Similarly, C_d is the ratio of the cost of transmitting a data message to the I/O cost of a read-write request. We now present the model for computing the cost of servicing a read and a write request respectively. We denote by $COST_{ALG}(q)$ the cost of servicing a request q with an algorithm ALG. Given a request schedule ψ_o, the cost of servicing a request q, either a read or a write, is defined as follows:

Case A (Read request): Consider the request q as a read request $r_o^{p_i}$ from processor p_i for object o and let A_o be the allocation scheme of object o at this request. Then,

$$COST_{ALG}(r_o^{p_i}) = \begin{cases} 1 & \text{if } p_i \in A_o \\ 1 + C_c + C_d & \text{if } p_i \notin A_o \text{and } r_o^{p_i} \text{ is not a saving-read} \\ 2 + C_c + C_d & \text{if } p_i \notin A_o \text{and } r_o^{p_i} \text{ is a saving-read} \end{cases} \tag{2.1}$$

In Equation (2.1), if $p_i \in A_o$, for a read request q, object o is simply retrieved from p_i's local database. Otherwise, besides the I/O cost at the processor which outputs the object from its local database buffer, there is one C_c needed which is the cost of submitting a control message request to the server from CCU and one C_d needed which is the cost of transmitting the object from the server in A_o to processor p_i. From the above model, the only cost difference between a saving-read request and a non-saving-read request is one I/O cost. This is because a saving-read request need to save the object in the local database after the object is delivered to processor p_i.

Case B (Write request): Suppose that the request q is a write request $w_o^{p_i}$ and let A_o be the allocation scheme of object o at this request. Also, let A'_o be the allocation scheme

of object o after servicing this request. Note that A'_o contains t processors according to the *t-availability constraint*. Then, the cost of servicing this request is given by,

$$COST_{ALG}(w_o^{p_i}) = |A_o/A'_o| \cdot C_c + (|A'_o| - 1) \cdot C_d + |A'_o| \tag{2.2}$$

The explanation for the write cost is as follows. Each of write request creates a new version of object o. In order to keep the consistency of the object, an invalidate control message has to be sent to all the processor of (A_o/A'_o), at which the copies of the object o are obsolete. These processors are the processors of A_o(the old allocation scheme) that are not in A'_o(the allocation scheme after servicing this write request). Thus, this is the first term in the write-cost equation. The next part $(|A'_o| - 1) \cdot C_d$ is the cost of transferring the new copy of object from processor p_i to all the processors in new allocation scheme A'_o except p_i. The last part accounts for the I/O cost when processors in A'_o save the object into their local database. Fig.2.2 shows different allocation schemes of object j after serving the write request w_j^i in two situations, $i \in F \cup \{p\}$ and $i \notin F \cup \{p\}$.

For the whole request schedule $\psi_o = o_1 o_2 ... o_n$ and an initial allocation scheme IA_o, where o_i is either a read or a saving-read, or a write request, we define the *cost* of the request schedule ψ_o, denoted by $COST(IA_o, \psi_o)$, to be the sum of all costs of the read-write requests in the schedule, i.e.,

$$COST(IA_o, \psi_o) = \sum_{i=1}^{n} COST(o_i) \tag{2.3}$$

2.2.3 Definitions, terminologies and notations

In this section, we present some preliminary definitions and notations based on the introduction above. These definitions and notations will be used frequently throughout this thesis.

Table 2.1 presents a glossary of these frequently used definitions and notations.

2.3 Concluding Remarks

In this chapter, we introduced distributed database systems and some concepts widely used in data allocation and replication in distributed database. We also introduced the basic cost computation model that is adopted in the Distributed Object Management literature. Some important notations and definitions that will be used frequently in the rest of the thesis were introduced.

C_{io}	I/O cost of inputting (outputting) the object from (to) the local database.
C_c	Communication cost of transmitting a control-message.
C_d	Communication cost of transmitting a data-message.
$r_o^{p_i}$	A read request issued by processor p_i for object o.
$w_o^{p_i}$	A write request issued by processor p_i for object o.
$COST_{ALG}(r_o^{p_i})$	Cost of servicing a read request $r_o^{p_i}$ with an algorithm ALG.
$COST_{ALG}(w_o^{p_i})$	Cost of servicing a write request $w_o^{p_i}$ with an algorithm ALG.
ψ_o	A request schedule for object o.
$\psi_o(i)$	The ith request sub-schedule for object o.
$F \bigcup \{p\}$	Server set F and the floating processor p, the composition of an initial allocation scheme.
t	Minimum number of copies of the object that must exist in the system, which is referred to as *t-availability constraint*.
A_o	The allocation scheme of object o at a request q for object o.
A'_o	The allocation scheme of object o after servicing the request q for object o.
$TypeI$	A sub-schedule consisting of only read requests before the first write request in a request schedule ψ_o for object o, denoted by ψ_o^{ss1}.
$TypeII$	A sub-schedule consisting of a write request followed by zero or more read requests which come before the next write request in the request schedule ψ_o for object o, denoted by ψ_o^{ss2}.
$COST_{ALG}(IA_o, \psi_o)$	The cost of servicing a request schedule ψ_o with an algorithm ALG with the initial allocation scheme IA_o.
τ	The Length of an initial request schedule, i.e., the number of requests in an initial request schedule.

$N_p(i)$	The set of different reading processors (processors issuing read requests) of the ith sub-schedule $\psi_o(i)$ which do not belong to the initial allocation scheme in Type I sub-schedule, or do not belong to the allocation scheme after servicing the first write request in Type II sub-schedule.
$n_p(i)$	The number of reading processors of $N_p(i)$. We have $\|N_p(i)\| = n_p(i)$.
$N_a(i)$	The set of all different reading processors of the ith sub-schedule $\psi_o(i)$.
$n_a(i)$	The number of all reading processors of $N_a(i)$. We have $\|N_a(i)\| = n_a(i)$.
$n_r(i)$	The total number of read requests in the ith sub-schedule.
$n_r(i,k)$	The total number of read requests issued by processor k in the ith sub-schedule.

Table 2.1: Glossary of definitions and notations

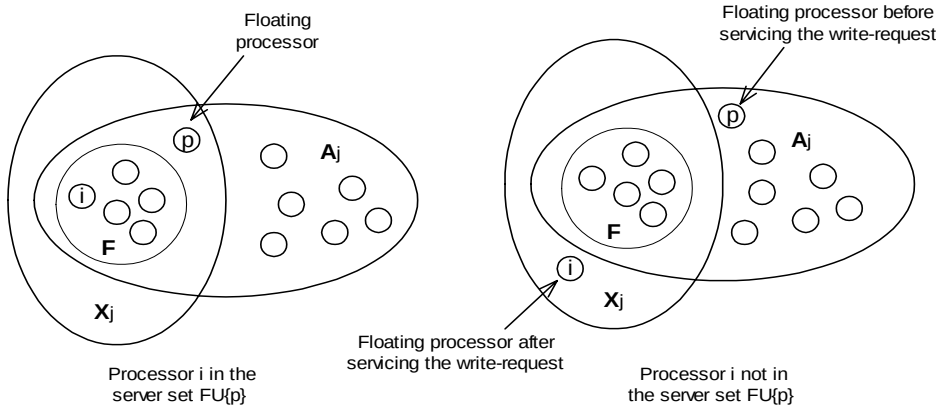


Figure 2.2: Different allocation scheme according to a write-request w_j^i when server $i \in F \cup \{p\}$ and $i \notin F \cup \{p\}$

Chapter 3

Data Allocation and Replication with Finite-size Buffer Constraints

Formally, the distributed system is a set of m processors with finite-size buffer constraint interconnected via a message-passing network and a CCU(central control unit). The CCU is aware of the entire allocation scheme of every object and every processor knows whether or not the copy of the object is available in its local database. As we have mentioned before, the local database at a processor is a set of objects at this processor. All read-write requests issued by processors for objects arrive at CCU in a concurrent fashion and are serialized by some CCM. Note that with our definitions in previous chapter, we assume a homogeneous system where the communication cost of a control-message between any pair of processors, the communication cost of a data-message between any pair of processors and the I/O cost is identical at all the processors.

In order to service the requests in a minimum cost, at first, CCU invokes the competitive data allocation algorithm, referred to as *Dynamic Window Mechanism*(DWM), to service the requests and to determine the allocation scheme of the object at any point in time. As we consider the scenario in which processors have local database buffer of finite size, we are

naturally confronted with a problem when a processor's local database buffer is full while this processor is servicing a saving-read request determined by DWM or a write request. Apparently, in order to gain a minimum total cost of service, this processor needs to save the copy of the object of such requests in its local database. Then object replacement strategies need to be invoked to decide which object should be purged out of the processor's local database to make room for new object copy, which has the least effect of later service.

In Section 3.1, we present the DWM algorithm, in Section 3.2, we present three types of object replacement strategies to deal with object replacement problem. We also give out the cost computation functions according to different strategies. In a general sense, we consider the above mentioned scenario in which the object sizes are assumed to be equal. We also consider the situation where the object sizes are different from each other.

3.1 DWM Algorithm

Dynamic Window Mechanism(DWM) adapts to the changes in the patterns of read-write requests. This dynamic algorithm uses a windowing mechanism. Requests to the distributed database system arrive from several processors and CCM in CCU serializes them in such a way that it outputs at most one request in each time unit σ (without losing generality, $\sigma = 1$) [12, 17, 24, 26]. Then CCU invokes the DWM algorithm to serve these requests in a cost-effective manner. The DWM algorithm considers requests in batches, each batch comprising at most τ requests and forms an initial request schedule. For example, this request batch could be, $\psi = r_{j_1}^{p_1} r_{j_2}^{p_2} w_{j_3}^{p_3} r_{j_4}^{p_4} w_{j_5}^{p_5} w_{j_6}^{p_6} ... r_{j_k}^{p_n}$, where $j_1 j_2 ... j_k$ are the requested objects and $p_1 p_2 ... p_n$ are the processors issuing these requests. The number of requests is at most τ. Fig.3.1 shows the arrival of requests to DWM after the processing of CCM and shows how batches of requests for servicing are considered by DWM.

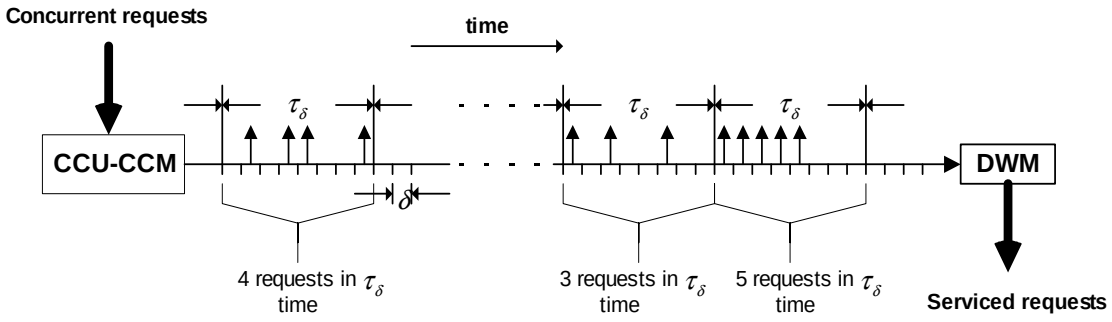


Figure 3.1: Concurrent control mechanism of CCU

Firstly, the request windows are formed by selecting the requests for the same object when DWM algorithm considers each request one after another in ψ. As an example, $win(j_k) = (r_{j_k}^{p_1} r_{j_k}^{p_2} w_{j_k}^{p_3} ... r_{j_k}^{p_n})$ could be one request window where all the requests are aimed at object j_k. After a request window $win(j_k)$ is formed, DWM initializes the allocation scheme for object j_k and considers requests in $win(j_k)$ from the beginning and two kinds of sub-schedules are generated. For example, $\psi_{j_k}(i) =< r_{j_k}^{p_1} r_{j_k}^{p_2} >$ is a sub-schedule of $win(j_k)$. These operations are presented as follows and Fig.3.2 illustrates the DWM algorithm.

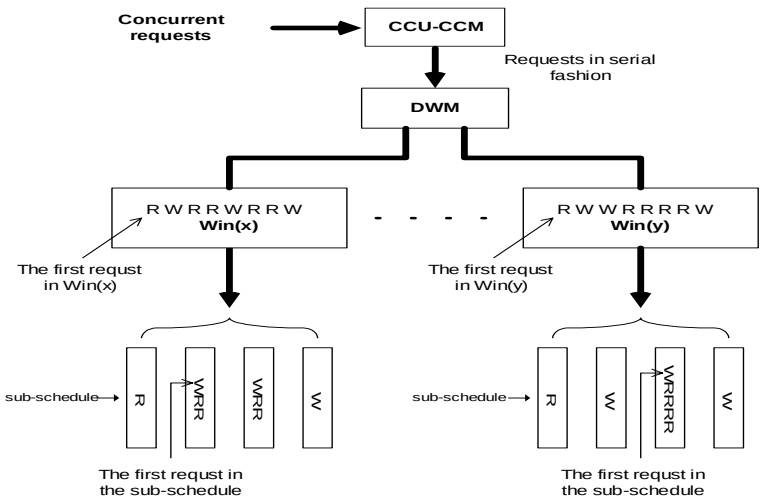


Figure 3.2: DWM working style

Procedure: **Sub-schedule generation**

Step 1. Create a new $\psi_o(i)$ for object o and insert the current request into $\psi_o(i)$. If the request is the last request in $win(o)$, go to Step 3; otherwise, consider the next request and go to Step 2.

Step 2. If the current request is a read request, we simply insert the request into $\psi_o(i)$. If this read request is the last request in $win(o)$, go to Step 3; otherwise, consider the next request and go to Step 2. If the current request is a write request, service the requests in $\psi_o(i)$, and go to Step 1.

Step 3. Service the requests in $\psi_o(i)$ and end the request window operation for object o.

After the sub-schedule generation, there may be several but finite number of sub-schedules for $win(o)$. Each sub-schedule $\psi_o(i)$ belongs to either Type I or Type II sub-schedule. The definition and how DWM algorithm serves the Type I and Type II sub-schedules respectively are described as follows.

1. Type I sub-schedule

Type I sub-schedule for object o, denoted by ψ_o^{ss1}, is derived from $win(o)$ consisting of only read requests before the first write request comes up in the same order in $win(o)$. It is noticed that If the Type I sub-schedule does exit, it must be presented in the beginning of $win(o)$. So it is defined that $\psi_o^{ss1} = \psi_o(1)$. There is a definition:

$$\Delta = \Big[n_r(1) - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k) - n_p(1)\Big] \cdot (C_c + C_d) \tag{3.1}$$

$N_p(1)$ is the set of reading processors (issuing read requests) which do not belong to the initial allocation scheme in Type I sub-schedule $\psi_o(1)$. $n_p(1)$ is the number of processors which fall into $N_p(1)$. $N_a(1)$ refers to the set of processors issuing read requests in sub-schedule $\psi_o(1)$,

no matter if these processor belong to or do not belong to the allocation scheme. $n_r(1)$ is the total number of read requests in sub-schedule $\psi_o(1)$. $n_r(1,k)$ refers to the total number of read requests issued by a processor k in sub-schedule $\psi_o(1)$. C_c and C_d represent the cost of transmitting an control message and the cost of transmitting an data message respectively. Referring to Equation (2.1), if all the first read requests issued by different processors in $N_p(1)$ are considered as saving-read requests, then the cost of servicing read requests in Type I sub-schedule ψ_o^{ss1} is less by Δ (given by (3.1)) than that when these requests are not considered as saving-read requests. However, if these requests are considered as saving-read requests, then it incurs $n_p(1)$ amount of I/O cost more than that of the cost when read requests are not considered as saving read requests. Besides, a write request in $\psi_o(2)$ following $\psi_o(1)$, if it exists, incurs at most $n_p(1)C_c$ control-message cost more than that when the reads are not considered as saving-read requests. This part of cost comes from the consideration that the write is to "invalidate" the redundant replicas existed in the system. Therefore, if the DWM finds that $\Delta > \Big[n_p(1) + n_p(1) \cdot C_c\Big]$, each first of requests issued by each processor in $N_p(1)$ should be dealt with as a saving-read request. Otherwise, if $\Delta \leq \Big[n_p(1) + n_p(1) \cdot C_c\Big]$, all the requests should be considered as non-saving-read requests.

2. Type II sub-schedule

Type II sub-schedule, denoted by ψ_o^{ss2}, is extracted from $win(o)$ consisting of a write-request at the very beginning followed by all the read requests until the next write-request appears in the $win(o)$. It must be noticed that the order of sub-schedule is the same as that in the original request sequence in the request window $win(o)$. The operation of DWM almost resembles what it does to Type I sub-schedule. It is defined as follows:

$$\Delta = \Big[n_r(i) - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k) - n_p(i)\Big] \cdot (C_c + C_d) \qquad (i \neq 1) \qquad (3.2)$$

$N_p(i)$ means the set of processors issuing read requests in the ith sub-schedule which do not belong to the allocation scheme formed after the first write request is serviced in Type II sub-schedule. $n_p(i)$ is the number of processors which belong to $N_p(i)$. $N_a(i)$ refers to all processors issuing read requests in the ith sub-schedule. $n_r(i)$ is the number of total read requests in the ith sub-schedule. $n_r(i,k)$ refers to the number of read requests issued by a processor k in the ith sub-schedule. If every first request issued by every processor in $N_p(i)$ is considered as a saving-read request, the service cost is less by Δ (given by (3.2)) than that when those requests are considered as non-saving-read requests. But, in this case as we describe in Type I sub-schedule, considering every first read request issued by different processors in $N_p(i)$ as a saving-read request incurs at most $\Big[n_p(i) + n_p(i) \cdot C_c\Big]$ amount of cost more than that when these requests are not regarded as saving-read requests (as seen in the analysis of Type I sub-schedule). Therefore, if the DWM finds that $\Delta > \Big[n_p(i) + n_p(i) \cdot C_c\Big]$, each first request issued by each processor in $N_p(i)$ should be treated as a saving-read request. Otherwise, if $\Delta \leq \Big[n_p(i) + n_p(i) \cdot C_c\Big]$, all the requests should be considered as non-saving-read requests.

3.2 Strategies To Deal With Object Replacement

In DWM algorithm read requests are serviced as saving-read requests and non-saving read requests, respectively. A saving-read request lets a processor save a copy of that object in its local database. If the distributed system is read extensive and it happens that many of these requests are considered by a dynamic data distribution algorithm as saving-read requests, there may be a situation that the local database of a processor may overflow. When the

quota of a processor's local database meets with the situation of overflow and a new object need to be stored in that processor, a decision must be made through a kind of replacement strategy, which decides whether or not the new object should be saved and if the new object is ordered to be saved, then which object in use should be picked and purged from the local database to make space for the new object. The same situation also need to be considered when processors are servicing write requests. Specifically, we are interested to study on the influence of most commonly used and popular replacement algorithms and its variants. Below we introduce these replacement algorithms and their variants that are of our interest. In order to have an all-sided view on this problem, we devise two kinds of environment models. One is Homogeneous Object Sizes model(Hom). The other one is Heterogeneous Object Sizes model(Het). We also introduce the updated cost functions in this section when some object replacement strategies are invoked. Thus, we study the influence of replacement strategies on the performance of DWM algorithm.

3.2.1 Two models

We implement and evaluate different object replacement strategies under two models. The two models are presented as follows:

Model A(Homogenous Object Sizes): In this scenario, we assume that all objects in the distributed system we proposed are of same size, $|O_i| = S$, where $|O_i|$ is denoted as the size of the object i and S is a constant. In this case, the object replacement algorithms are much simpler, because when replacement is needed, only one qualifying object is considered by replacement algorithm in a processor's local database. We normalize the object size to 1. Three replacement algorithms, namely No Replacement(NR), Least Recently Used(LRU) and Least Frequently Used(LFU), are implemented for comparison purposes.

Model B(Heterogenous Object Sizes): In this scenario, we assume that the sizes of objects in the distributed system are different, i.e., $|O_i| = S_j(j = 1, 2, \ldots, n)$, where $(S_1, S_2, \ldots, S_n)$ account for a range of possible object sizes. In this case, the object replacement algorithms are more complicated, since in order to make enough space for a new object, it is possible that more than one object in local database may need to be evicted by the replacement algorithms. Thus, except No Replacement, we design two other strategies which are the extensions of LRU and LFU respectively, referred to as Heterogenous object sizes LRU(LRU^{het}) and Heterogenous object sizes LFU(LFU^{het}). These object replacement algorithms will be introduced in next subsection.

3.2.2 Object replacement algorithms

It is a well known fact that replacement algorithms improve the cache performance. The choice and performance of a replacement policy depends on its ability to purge out an object from memory that has a low probability of near future access thus making room for a new object. In order to achieve a better performance, strategies need to explore the principles of spatial and temporal locality of reference [16, 35]. Spacial locality implies that if an object is referenced, then nearby objects will be more likely be referenced while temporal locality implies that a recently referenced object will tend to be referenced again in the near future. In fact, replacement algorithms have been widely used in many other related areas such as memory paging system [16], database disk buffering [36] and web proxy cache [37], to quote a few.

The different strategies used in Model A are discussed below.

Strategy I: No Replacement(NR): Intuitively, there is no replacement taking place when a processor's local database is full. When a processor q services the saving read request and

the write request issued by q, the copy of object will not be saved in q's local database. As far as a write request is concerned, processor q will also issue the new version of the object to the current set of F servers and the floating processor p in order to satisfy the *t-availability constraint* and obsolete copies will be invalidated as usual to maintain the object consistency. This strategy gives us the base performance and, obviously, any strategy that performs worse than this strategy is not worth considering.

Strategy II: Least recently used(LRU): The LRU algorithm is one of the most popular replacement policies, particularly in commercial implementations. It is used widely in database buffer management and file systems. This policy capitalizes on the principle of temporal locality and purges the object used least in the recent past on the assumption that it will not be referenced in the near future.

We describe here briefly on the workings of LRU with respect to DWM algorithm in the sense that we explicitly take into account the t-availability constraint in our implementation. Let $Set(t)$ be the *resident set* of all objects in a server at time t. Let $r(t)$ be the object to be replaced from $Set(t)$ at time t. We define a backward distance $b_t(i)$ [16] as the number of time slots from time t to the most recent reference of an object i in the past. The LRU replaces the object in $Set(t)$ which has the longest backward distance. For example, if $b_t(j) = max_{i \in Set(t)}\{b_t(i)\}$, then $r(t) = j$. This policy is simple to implement but becomes computationally expensive for large object resident set. It is important to note that the *t-availability* constraint must be satisfied, which is $t^{ini} > 1$, where t^{ini} is the initial number of copies of the object in the distributed system. In order to keep at least t^{ini} copies of an object existing in the distributed system, we introduce another parameter, namely $l\_flag(i)$ (a flag, which indicates whether the current copy is one of the last t^{ini} copies of an object i available in the system) which has two states represented by 0 and 1. If $l\_flag(i) = 0$, it means that this copy of object i is one of the last t^{ini} copies in the entire distributed system.

Then, we will restrict LRU not to consider this copy as a legal candidate of invalidation. If $l\_flag(i) = 1$, then LRU will work as per the above description and may or may not replace this object. The update of $l\_flag(i)$ is done by CCU, as CCU knows the entire allocation scheme of all the objects in the system.

Strategy III: Least frequently used(LFU): This algorithm uses the history of references to predict the probability of a future access. The object stream maintains a reference count for all its objects and replaces the object that has been used least frequently. Let $Set(t)$ be the *resident set* of all objects in a server and $r(t)$ be the object to be replaced from $Set(t)$ at time t. Then $r(t)$ is the object in $Set(t)$ which has been least referenced in the past. However, the LFU algorithm has an obvious weakness, namely the fact that objects with large frequency counts tend to remain in the resident set $Set(t)$, no matter if these objects will be accessed again or not, which prevents the newly joined objects from gathering enough reference counts to stay in the resident set. This causes the so called *cache pollution* phenomenon [35], in which inactive data tends to reduce the performance of the LFU algorithm. Also, as in the case of LRU, here too we consider the *t-availability* constraint while replacing an object by LFU, by using the status flag l_flag in our implementations.

As we have introduced in the previous subsection, for Model B, we devise two other object replacement strategies. Strategy I used in Model B still uses "No Replacement", which has no difference used in both Model A and Model B. We present the other two cases under Strategy II and III as follows.

Heterogenous object sizes LRU(LRU^{het}): We devise this replacement algorithm exclusively to deal with the situation wherein objects are of different sizes. In fact, our *size-based* version of LRU (LRU^{het}) is a logical extension of LRU. This policy also exploits temporal locality of reference, keeping the recently used objects while dropping the least recently used objects. However, when purging a selected object, LRU^{het} may choose more than one object

to be purged because of the possibility that the size of first chosen object from the resident set may be smaller than the new object that will be saved in resident set. For LRU$^{\text{het}}$, besides a backward distance $b_t(i)$ defined above, the object size, denoted by $size(i)$, for object i is also defined to record each object i's size. Thus, the LRU$^{\text{het}}$ has to maintain both the $b_t(i)$ and $size(i)$ information for every object. Let $size_t(j)$ be the size of new object j to be saved in the memory at time t and $sp(t)$ be the available space at time t. The process of LRU$^{\text{het}}$ is as follows. First, LRU$^{\text{het}}$ chooses an object with the largest $b_t(i)$ and $l\_flag(i) = 1$(satisfying *t-availability* constraint); let the object to be purged be u_1. Then, LRU$^{\text{het}}$ checks if $\Big[size(u_1) + sp(t)\Big] \geq size_t(j)$; if available space is enough, then purge the object u_1 and save object j; if $\Big[size(u_1) + sp(t)\Big] < size_t(j)$, LRU$^{\text{het}}$ has to find the next largest $b_t(i)$, for example, $b_t(u_2)$, satisfying the condition $l\_flag(i) = 1$ and see if free space is enough or not; the iteration in above process continues until enough room is made available for new object j, and we purge all the objects $u_1, u_2, u_3, ..., u_r$ and save the object j into the resident set $Set(t+1)$. Finally, LRU$^{\text{het}}$ initializes $b_{t+1}(j)$ and $l\_flag(j)$ for object j and updates $sp(t+1)$ by $sp(t+1) = \Big[\sum_{k=1,2,3,...,r} size(u_k) + sp(t) - size(j)\Big]$ and the related $l\_flag(v)(v = u_1, u_2, u_3, ..., u_r)$ as well. Table 3.1 summarizes the workings of LRU$^{\text{het}}$.

Heterogenous object sizes LFU(LFU$^{\text{het}}$): This algorithm is also designed exclusively to deal with the situation wherein objects are considered of different sizes. Similar to LRU$^{\text{het}}$, our *size-based* version of LFU(LFU$^{\text{het}}$) is the extension of LFU, using the object popularity history to predict the probability of a subsequent reference. The object chain maintains a reference count for all objects in the local buffer and replaces the object that has been used least frequently. We denote $C_t(i)$ as the reference count of object i and let $Set(t)$ be the *resident set* of all objects in a server at time t. The object size ($size(i)$) is introduced to record the size of the object i. LFU$^{\text{het}}$ may choose more than one object to be purged in order to make enough space because of the possibility that the size of first chosen object to

Algorithm LRU$^{\text{het}}$:

For each new object j do

If j is in resident set $Set(t)$ then update $b_t(j)$

else while there is not enough free slots for j

Find all objects with the largest $b_t(q)(q \in Set(t))$ and whose $l\_flag = 1$

Evict all such objects q which satisfy $[sp(t) + \sum size(q)] \geq size(j)$

Save j

Update $b_{t+1}(j)$, $l\_flag(p)$, $sp(t+1)$ and $l\_flag(q)$

Table 3.1: Pseudo code of LRU$^{\text{het}}$ algorithm

be replaced in resident set may be smaller than the new object that will be saved in resident set. Let $size_t(j)$ be the size of new object j to be saved in the buffer at time t and $sp(t)$ be the available space which currently the resident set $Set(t)$ holds at time t. The process of LFU$^{\text{het}}$ is as follows. LFU$^{\text{het}}$ first chooses an object in $Set(t)$ which has been least referenced in the past, say minimum $C_t(i)$; also because of *t-availability* constraint, only objects with $l\_flag(i) = 1$ are qualified to be chosen; let the object to be purged be u_1. LFU$^{\text{het}}$ then checks if $\Big[size(u_1) + sp(t)\Big] \geq size(j)$; if available space is enough, then purge the object u_1 and save object j; if $\Big[size(u_1) + sp(t)\Big] < size_t(j)$, LFU$^{\text{het}}$ will find the minimum $C_t(i)$ next to $C_t(u_1)$, for example, $C_t(u_2)$, satisfying the same condition $l\_flag(i) = 1$ and check if free space is enough or not; the iteration in above process continues until the resident set has enough space for new object j, then we purge all the objects selected, say $u_1, u_2, u_3, ..., u_r$ and save the object j into the resident set $Set(t+1)$. LFU$^{\text{het}}$ finally initializes $C_{t+1}(j)$ and $l\_flag(j)$ for object j and updates the $sp(t+1)$ by $sp(t+1) = \Big[\sum_{k=1,2,3,...,r} size(u_k) + sp(t) - size(j)\Big]$. If

needed, change the state of $l\_flag(v)(v = u_1, u_2, u_3..., u_r)$. Table 3.2 summarizes the workings of LFU$^{\text{het}}$.

Algorithm LFU$^{\text{het}}$:

for each newer object j do

If j is in resident set $Set(t)$ then update $C_t(j)$

else while there is not enough free slots for j

Find the objects with the minimum $C_t(q)(q \in Set(t))$ and $l\_flag = 1$

Evict all such objects q which satisfy $[sp(t) + \sum size(q)] \geq size(j)$

Save j

Update $C_{t+1}(j)$, $l\_flag(j)$, $sp(t+1)$ and $l\_flag(q)$

Table 3.2: Pseudo code of LFU$^{\text{het}}$ algorithm

3.3 Modified Cost Function

The performance metric in our study is the cumulative cost of servicing all kinds of requests. It is noticed that the extra cost need to be calculated when the object replacement happens in the local database. Because any replacement action requires a server to invalidate one or more copies of the objects in its local buffer and directly leads to the change of the allocation schemes of evicted objects, this kind of replacement has to be reported to CCU as CCU unit knows the allocation scheme of every object. Hence, the additional control-message costs has to be added. This situation happens only when DWM handles saving read requests and write requests. The cost calculation is, thus, affected by different object replacement strategies. We

analyze such impacts according to different strategies in different models. From the definition of Strategy II and Strategy III in Model A, we find that the additional cost evoked by an object replacement action is identical for these two strategies. The reason is easy to understand. When local database is full, both Strategy II and Strategy III pick up and purge an object in use to make space for a new object copy. The only difference comes from their different ways of picking up an object eviction candidate. Thus, we can classify the cost functions in two categories. We define *DWM-No Replacement* as the algorithm to service the request schedule when Strategy I is used when a processor's local database is full. We define *DWM-Replacement* as the algorithm to service the request schedule when Strategy II and Strategy III are used with DWM respectively.

Given a request schedule, suppose that q is a request in the schedule. Let A_o be the allocation scheme at q. Also, let A'_o be the allocation scheme after request q is serviced. The cost of request q is denoted by $COST_{ALG}(q)$, where ALG indicates what algorithm is used to service the request schedule. We then present the cost function in terms of different ALG, namely *DWM-No Replacement*, denoted by *DWM-NR*, and *DWM-Replacement*, denoted by *DWM-R*, under Model A and Model B respectively.

1. The cost function in Model A is updated as follows:

   As in Section 2.2.2, we also consider the service cost in two cases. In Case A, we present the cost of servicing a read request $r_o^{p_i}$ and in Case B, we present the cost of servicing a write request $w_o^{p_i}$.

   Case A: Consider a request q as a read request $r_o^{p_i}$ from processor p_i for object o. Then, the servicing cost by DWM-No Replacement is given by,

$$COST_{DWM-NR}(r_o^{p_i}) = \begin{cases} 1 & \text{if } p_i \in A_o \\ 1 + C_c + C_d & \text{if } p_i \notin A_o \text{ and } r_o^{p_i} \text{ is not a saving-read} \\ 2 + C_c + C_d & \text{if } p_i \notin A_o \text{ and } r_o^{p_i} \text{ is a saving-read} \\ & \text{and } p_i\text{'s local database is not full} \\ 1 + C_c + C_d & \text{if } p_i \notin A_o \text{ and } r_o^{p_i} \text{ is a saving-read} \\ & \text{and } p_i\text{'s local database is full} \end{cases} \tag{3.3}$$

When the processor's local database is full, by strategy DWM-No Replacement, the processor does not save any new copy of the object in its local database. Observe that the cost of a saving read does not differ from that of a non-saving read under this situation.

Next, we present the servicing cost by DWM-Replacement, which is given by,

$$COST_{DWM-R}(r_o^{p_i}) = \begin{cases} 1 & \text{if } p_i \in A_o \\ 1 + C_c + C_d & \text{if } p_i \notin A_o \text{ and } r_o^{p_i} \text{ is not a saving-read} \\ 2 + C_c + C_d & \text{if } p_i \notin A_o \text{ and } r_o^{p_i} \text{ is a saving-read} \\ & \text{and } p_i\text{'s local database is not full} \\ 2 + 2 \cdot C_c + C_d & \text{if } p_i \notin A_o \text{ and } r_o^{p_i} \text{ is a saving-read} \\ & \text{and } p_i\text{'s local database is full} \end{cases} \tag{3.4}$$

When p_i's local database is not full, no object replacement happens; when the processor's local database is full, by strategy DWM-Replacement, the processor picks up and evicts an object from local database to make space for the newly inputting object. Thus, an extra cost of control-message (C_c) is needed to inform the CCU of the change of the allocation scheme of the object that is evicted from processor p_i's local database buffer.

Case B: Consider a request q as a write request $w_o^{p_i}$ from processor p_i for object o. Then, the servicing cost by DWM-No Replacement is given by,

$$COST_{DWM-NR}(w_o^{p_i}) = \begin{cases} |A_o/A'_o| \cdot C_c + (|A'_o| - 1) \cdot C_d + |A'_o| \\ \qquad\qquad \text{if } p_i\text{'s local database is not full} \\ (|A_o/A'^*_o| - 1) \cdot C_c + |A'^*_o| \cdot C_d + (|A'^*_o| - 1) \\ \qquad\qquad \text{if } p_i\text{'s local database is full} \end{cases} \tag{3.5}$$

Regarding a write request, when p_i's local database is full, the new version of the object needs to be disseminated to all the processors in A'^*_o, which is different from A'_o. The cost of transmitting the write object to A'^*_o is higher (by one C_d) than that of previous term as A'^*_o consists of t processors not including processor p_i. Observe also that, due to the same reason, the cost of a write request when i's local database is full is less by one C_c and one I/O cost than previous term.

Next, we present the servicing cost by DWM-Replacement, which is given by,

$$COST_{DWM-R}(w_o^{p_i}) = \begin{cases} |A_o/A'_o| \cdot C_c + (|A'_o| - 1) \cdot C_d + |A'_o| \\ \qquad\qquad \text{if } p_i\text{'s local database is not full} \\ |A_o/A'_o| \cdot C_c + (|A'_o| - 1) \cdot C_d + |A'_o| + C_c \\ \qquad\qquad \text{if } p_i\text{'s local database is full} \end{cases} \tag{3.6}$$

When p_i's local database is not full, no object replacement happens; when the processor's local database is full, by strategy DWM-Replacement, besides that the writing object needs to be transmitted to other processors to form a new allocation scheme and obsolete copies of the object need to be invalidated, the processor also need to pick up and evicts an object from local database to make space for the newly inputting object.

Thus, an extra cost of control-message is needed to inform the CCU of the change of the allocation scheme of the object that is evicted from processor p_i's local database buffer. Thus, one more C_c is needed.

2. The cost function in Model B is updated as follows:

After we update the cost calculation of requests in Model A, we define the cost functions used in Model B. According to the definition of object replacement Strategy I, we notice that DWM-No Replacement works the exactly same way in Model B as it does in environment Model A. When the processor's local database does not have available buffer space to store the new copy of object, the processor simply gives up doing so. Thus, the cost function by DWM-No Replacement is obtained from Equation (3.3) for servicing a read request and seen from Equation (3.5) for servicing a write request. Then, we only consider the cost function of requests by strategy of DWM-Replacement. As the definition above, the $COST_{ALG}(q)$ is defined in two cases as in Model A as follows:

Case A: Consider a request q as a read request $r_o^{p_i}$ from processor p_i for object o. Then, the servicing cost by DWM-Replacement is given by,

$$COST_{DWM-R}(r_o^{p_i}) = \begin{cases} 1 & \text{if } p_i \in A_o \\ 1 + C_c + C_d & \text{if } p_i \notin A_o \text{ and } r_o^{p_i} \text{ is not a saving-read} \\ 2 + C_c + C_d & \text{if } p_i \notin A_o \text{ and } r_o^{p_i} \text{ is a saving-read} \\ & \text{and } p_i\text{'s local database is not full} \\ 2 + C_c + C_d + m \cdot C_c & \text{if } p_i \notin A_o \text{ and } r_o^{p_i} \text{ is a saving-read} \\ & \text{and } p_i\text{'s local database is full} \end{cases} \tag{3.7}$$

Case B: Consider a request q as a write request $w_o^{p_i}$ from processor p_i for object o. Then, the servicing cost by DWM-Replacement is given by,

$$COST_{DWM-R}(w_o^{p_i}) = \begin{cases} |A_o/A'_o| \cdot C_c + (|A'_o| - 1) \cdot C_d + |A'_o| \\ \qquad\qquad \text{if } p_i\text{'s local database is not full} \\ |A_o/A'_o| \cdot C_c + (|A'_o| - 1) \cdot C_d + |A'_o| + m \cdot C_c \\ \qquad\qquad \text{if } p_i\text{'s local database is full} \end{cases} \tag{3.8}$$

The reasoning for these cost functions are identical to those in Model A. Observe that the extra cost of control-message is $m \cdot C_c$ compared to that of exact one C_c in Model A. This is because that both LRU$^{\text{het}}$ and LFU$^{\text{het}}$, acting as object replacement strategies, possibly need to pick up m objects to evict in order to make enough space for the newly inputting object.

Chapter 4

Analysis of the Data Allocation Algorithm With Finite-size Buffers

In this chapter, we use competitive analysis to quantify performance of our proposed DOM algorithms with finite-size buffer constraints. We consider a request $win(o)$ of an object o. The cost of servicing all the requests in a given request schedule ψ is the sum of the individual cost of servicing each request window for the respective requested objects. The cost calculation functions are presented in previous chapter. The cost of servicing requests in $win(o)$ is given by,

$$COST_{ALG}(IA_o, \psi_o) = \sum_{i=1}^{k} COST_{ALG}(IA_o(i), \psi_o(i)) \tag{4.1}$$

where $\psi_o(i)$ is the ith sub-schedule with an initial allocation scheme $IA_o(i)$. We assume that IA_o in $win(o)$ is $F \bigcup \{p\}$, in which F is a set of $(t-1)$ processors and p is the floating processor (t-*available* constrained). ALG represents different DOM algorithms with buffer capacity constraints.

4.1 Competitiveness

In the world of online computation, an algorithm must produce a sequence of decisions that will have an impact on the final quality of its overall performance. Each of these decisions must be made based of past events without secure information about the future. Such an algorithm is called an *online algorithm*. The traditional approach to study online algorithms falls within the framework of distributional (or average-case) complexity, whereby one hypothesizes a distribution on events sequences and studies the expected total cost or expected cost per event. During the past 10 years the analysis in this subject has been renewed largely as a result of the approach of *competitive analysis* [8]. The roots of competitive analysis can be found in classical combinatorial optimization theory [5] and in the analysis of data structure. The competitive analysis measures the quality of an online algorithm on each input sequence by comparing its performance to that of an optimal *offline algorithm*, which is an unrealizable algorithm that has full knowledge of the future. Competitive analysis thus falls within the framework of worst-case complexity.

4.1.1 Offline and online dynamic data allocation algorithms

We begin with a discussion of the concept of an offline dynamic data allocation algorithm and an online dynamic data allocation algorithm.

An offline dynamic allocation algorithm is an optimal and ideal algorithm, denoted by OPT. It knows the whole request schedule in advance, and maps the request schedule to a configured execution schedule before servicing the requests. An optimal offline algorithm OPT is such that for all legal request schedules, the cost of servicing schedules by OPT is minimum.

An online dynamic data allocation algorithm does not have knowledge of the whole schedule, it changes the replication and allocation scheme based on the previous request received. The

complication inherent in online algorithms is that each online input (a request) influences the cost of the overall solution. Upon receiving a request o_i, an online dynamic allocation algorithm (denoted by ALG) configures the next allocation scheme A_i based on the previous configured schedule and the current request o_i. In other words, the algorithm ALG configures the associated allocation scheme of an object immediately after the request issued for this object is serviced, while before the next request is serviced.

4.1.2 The competitive ratio and competitiveness

Competitiveness is a widely accepted way to measure the performance of an online DOM algorithm [4, 6, 7]. Intuitively, an online DOM algorithm is c-competitive if this algorithm, for any request schedule, costs at most c times as much as any other (online or offline) algorithms. Formally, a c-competitive dynamic allocation algorithm ALG is one for which there are two constants c and α, such that, for an arbitrary initial allocation scheme IA and an arbitrary request sequence ψ, $COST_{ALG}(IA, \psi) \leq c \cdot COST_{OPT}(IA, \psi) + \alpha$, where OPT is an offline data allocation algorithm that produces the minimum cost legal allocation schedule for any input request sequence.

When the addictive constant α is less than or equal to zero, we may say for emphasis that ALG is strictly c-competitive. Allowing a positive constant α reflects the view that for intrinsic online problems we have an arbitrarily long input sequence with unbounded cost. The constant α becomes insignificant as we consider longer initial subsequences. Moreover, even for finite input sequences, the use of the addictive constant α allows an performance ratio that does not depend on initial conditions.

If ALG is c-competitive, we say that ALG attains a competitive ratio c. An algorithm is called competitive if it attains a constant competitive ratio c. Although c may be a function of the problem parameters, it must be independent of the input request schedules. A

(strictly) c-competitive online algorithm ALG is an algorithm with the restriction that ALG must compute online. Thus, for each input request schedule, a c-competitive algorithm is guaranteed to incur a cost within a factor c of the optimal offline cost. We note that the competitive ratio is at least 1 and the smaller it is, the better ALG performs with respect to OPT.

We make no requirement or assumptions concerning the computational efficiency of a competitive online algorithm. In the more traditional offline complexity studies, we are primarily concerned with approximation algorithms that compute within polynomial time. Thus, strictly speaking, c-competitive online algorithms and polynomial time c-approximation algorithms are not comparable.

4.2 Competitive Ratio of Different Strategies

After formulating the cost function of dynamic data allocation algorithm with finite-size buffer constraints for an arbitrary schedule in the previous chapter, we can analyze them in terms of competitiveness. As the way we categorized the cost functions according as the different object replacement strategies implemented with DWM, say DWM-No Replacement (denoted by DWM-NR) and DWM-Replacement (denoted by DWM-R) algorithms, we show the competitive ratios of different dynamic data allocation algorithms with buffer constraints.

4.2.1 Competitive ratio of dynamic allocation DWM-No Replacement

In this section we present the competitive ratio of DWM-No Replacement. The cost function of DWM-No Replacement strategy was presented in previous chapter. In terms of our model, DWM-No Replacement considers a $win(o)$ of an object o in an arbitrary request schedule ψ.

Each request is serviced according to DWM as a non-saving read, a saving read or a write. During the servicing process, when the processors' local database is full, object replacement Strategy I (No replacement) does not pick and purge any object in use in the local database. The incoming copy of the object is just discarded instead of being saved into the local database. Note that DWM selects a set of processors F of size $(t-1)$ and a processor $p \notin F$, such that $F \bigcup \{p\}$ is the initial allocation scheme of size t.

Theorem 1 *For any integer $t \geq 1$, DWM-No Replacement algorithm is $(1 + C_c + C_d)$-competitive.*

we prove the following lemma before realizing the result of Theorem 1.

Lemma 1 *Suppose that $IA_o(i)$ is the initial allocation scheme of object o on $\psi_o(i)$ and $\psi_o(i) = \{\psi_o^{ss1}, \psi_o^{ss2}\}$ in $win(o)$. Suppose that $IA_o^*(i)$ is the initial allocation scheme of object o on $\psi_o(i)$ using an optimal algorithm that satisfies t-availability constraint. Then,*

$$COST_{DWM-NR}(IA_o(i), \psi_o(i)) \leq (1 + C_c + C_d) \cdot COST_{OPT}(IA_o^*(i), \psi_o(i))$$

Proof. We study two special types of sub-schedules, namely Type I sub-schedule ψ_o^{ss1} and Type II sub-schedule ψ_o^{ss2}. Type I sub-schedule consists of read requests only. Type II sub-schedule consists of a write request followed by zero or more read requests. We split the $win(o)$ into several subschedules, each one of which belongs to one of the two special types of sub-schedules mentioned above. It is noticed that if Type I sub-schedule exists, it must be at the beginning of $win(o)$. Then, we show that the cost of DWM-NR on both types of sub-schedules is at most $(1 + C_c + C_d)$ times as much as the cost of an optimum offline algorithm.

Proof of Part A (for read-only Type I sub-schedule): Suppose that Type I schedule $\psi_o(1) = r_o^{p_1} r_o^{p_2} r_o^{p_3} ... r_o^{p_n}$ comprises read requests, which come before the first write request,

where $1 \leq n \leq m$ and m is the number of processors in the distributed database system. Let $n_p(1)$ be the number of different processors issuing read requests (reading processors) of $\psi_o(1)$ which are not in $IA_o(1)$, say $F \bigcup \{p\}$. These processors form the processor set $N_p(1)$. To prove the Lemma 1, we analyze the cost bound comparison in the following two conditions.

Condition 1: Referring to the working style of DWM in Section 3.1, DWM may consider all the first read requests issued by the different processors in $\psi_o(1)$ which are not in $F \bigcup \{p\}$ as saving-read requests if $\Delta = \Big[n_r(1) - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k) - n_p(1)\Big] \cdot (C_c + C_d) > n_p(1) + n_p(1) \cdot C_c$ (refer to Equation (3.1)), where $n_r(1)$ is the total number of read requests in $\psi_o(1)$. Under the assumption that the processor's local database buffer has enough available space, the cost of servicing the read requests is given by,

$$\sum_{l=1}^{n} COST_{DWM-NR}(r_o^{p_l}) = n_p(1) \cdot (1 + C_c + C_d) + n_r(1) \tag{4.2}$$

The first read requests issued by different processors in $N_p(1)$ cost $(1 + C_c + C_d)$ more than all other read requests in $\psi_o(1)$. A cost of amount 1 accounts for the cost of saving I/O operation, i.e., saving the copy of object o in a processor's local database; C_c accounts for the cost of submitting the read request to a processor in the the initial allocation scheme $IA_o(1)$ of object o; C_d accounts for the cost of transmitting the object between any pair of processors. In addition, each read request, no matter if it is serviced locally or remotely, costs one unit for outputting the object o from a local database of a processor. Therefore, the cost of servicing all the read requests is $[n_p(1) \cdot (1 + C_c + C_d) + n_r(1)]$. Besides, as mentioned in Section 3.1, if DWM considers these read requests into saving-read requests, then this incurs additional more "invalidation cost" by at most $n_p(1) \cdot C_c$ than the cost when DWM does not consider these requests as saving read requests, since the future write request needs to invalidate all the obsolete copies of the object o. Thus, the total cost incurred by servicing $\psi_o(1)$ is,

$$\begin{aligned} COST_{DWM-NR}(IA_o(1), \psi_o(1)) &= \sum_{l=1}^{n} COST_{DWM-NR}(r_o^{p_l}) \\ &= n_p(1) \cdot (1 + C_c + C_d) + n_r(1) + n_p(1) \cdot C_c \\ &= n_r(1) + n_p(1) \cdot (1 + 2 \cdot C_c + C_d) \end{aligned} \tag{4.3}$$

Next we consider the assumption that all processors' local database buffer is full. Suppose that the worst case is that all the saving reads are not able to save the copy of object in processors' local database due to no available space at their local database buffer. In this case, all the read requests issued by different processors which are not in the initial allocation scheme are serviced remotely. Thus, we have,

$$\begin{aligned} COST_{DWM-NR}(IA_o(1), \psi_o(1)) &= \sum_{l=1}^{n} COST_{DWM-NR}(r_o^{p_l}) \\ &= \sum_{k \in N_a(1)/N_p(1)} n_r(1,k) + \Big[n_r(1) \\ &\quad - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k) \Big] \cdot (1 + C_c + C_d) \\ &= n_r(1) + \Big[n_r(1) \\ &\quad - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k) \Big] \cdot (C_c + C_d) \end{aligned} \tag{4.4}$$

There are $n_p(1)$ different reading processors which are not in the initial allocation scheme $IA_o(1)$. These processors issued $[n_r(1) - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k)]$ read requests, where $n_r(1)$ is the total number of read requests in $\psi_o(1)$ and $\sum_{k \in N_a(1)/N_p(1)} n_r(1,k)$ is the number of read requests issued by processor k (k is the processor of $N_a(1)$ that is not in $N_p(1)$). All these read requests issued by processors not in $IA_o(1)$ is serviced remotely and will cost $(1+C_c+C_d)$ more than the other reads. There are also $\sum_{k \in N_a(1)/N_p(1)} n_r(1,k)$ reads that are serviced locally.

Hence, the cost of servicing these local read requests is $\sum_{k \in N_a(1)/N_p(1)} n_r(1,k)$.

Now, we examine each item on the right hand side of Equation (4.3) and (4.4). Referring to Condition 1, $\Big[n_r(1) - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k) - n_p(1)\Big] \cdot (C_c + C_d) > n_p(1) + n_p(1) \cdot C_c$, we obtain,

$$\Big[n_r(1) - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k)\Big] \cdot (C_c + C_d) > n_p(1) \cdot (1 + 2 \cdot C_c + C_d) \tag{4.5}$$

Therefore, the upper bound of the cost in servicing the request schedule $\psi_o(1)$ due to an uncertainty of buffer space availability is given by,

$$\begin{aligned} COST_{DWM-NR}(IA_o(1), \psi_o(1)) \;\leq\; & n_r(1) + \Big[n_r(1) \\ & - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k)\Big] \cdot (C_c + C_d) \end{aligned} \tag{4.6}$$

After we obtain an upper bound of servicing cost of $\psi_o(1)$ by DWM-No Replacement, we give a lower bound of servicing cost by optimal offline algorithm OPT*. Note that we consider an OPT* algorithm as an algorithm without buffer constraint. Let us consider the same request schedule $\psi_o(1)$. Each read request costs at least one unit for the I/O cost to output the object from the local database, hence,

$$COST_{OPT^*}(IA_o^*(1), \psi_o(1)) \geq n_r(1) \tag{4.7}$$

Thus, from (4.6), we can see,

$$COST_{DWM-NR}(IA_o(1), \psi_o(1)) \;\leq\; n_r(1) + \Big[n_r(1)$$

$$
\begin{aligned}
& - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k) \Big] \cdot (C_c + C_d) \\
\leq \; & n_r(1) + n_r(1) \cdot (C_c + C_d) \\
= \; & n_r(1) \cdot (1 + C_c + C_d)
\end{aligned}
$$

Therefore, from (4.7) we conclude,

$$COST_{DWM-NR}(IA_o(1), \psi_o(1)) \leq (1 + C_c + C_d) \cdot COST_{OPT^*}(IA_o^*(1), \psi_o(1)) \tag{4.8}$$

Condition 2: Referring to Equation (3.1), if $\Delta = \Big[n_r(1) - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k) - n_p(1)\Big] \cdot (C_c + C_d) \leq n_p(1) + n_p(1) \cdot C_c$, then DWM-No Replacement should not consider the first read requests from different processors in $N_p(1)$ as saving read requests. In this case, processors will not save copy of objects in their local database. Then, we get the cost for servicing $\psi_o(1)$ by,

$$
\begin{aligned}
COST_{DWM-NR}(IA_o(1), \psi_o(1)) &= \sum_{l=1}^{n} COST_{DWM-NR}(r_o^{p_l}) \\
&= \sum_{k \in N_a(1)/N_p(1)} n_r(1,k) + \Big[n_r(1) \\
&\quad - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k)\Big] \cdot (1 + C_c + C_d) \\
&= n_r(1) + \Big[n_r(1) \\
&\quad - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k)\Big] \cdot (C_c + C_d)
\end{aligned} \tag{4.9}
$$

All the requests issued by the processors in $N_p(1)$ are serviced remotely. Each such read costs at least one for the I/O, one C_c for submitting the read request to a processor in the initial allocation scheme and one C_d for transmitting the object. The other read requests are

serviced locally, thus incurring one unit cost for I/O operation. The lower bound of servicing such a request schedule $\psi_o(1)$ is also given by $COST_{OPT^*}(IA_o^*(1), \psi_o(1)) \geq n_r(1)$, which is same as (4.7) in Condition 1.

Then, from (4.9), we can obtain,

$$\begin{aligned} COST_{DWM-NR}(IA_o(1), \psi_o(1)) &= n_r(1) + \Big[n_r(1) \\ &\quad - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k)\Big] \cdot (C_c + C_d) \\ &\leq n_r(1) \cdot (1 + C_c + C_d) \end{aligned} \tag{4.10}$$

Therefore, by comparing (4.10) and (4.7), we obtain,

$$COST_{DWM-NR}(IA_o(1), \psi_o(1) \leq (1 + C_c + C_d) \cdot COST_{OPT^*}(IA_o^*(1), \psi_o(1)) \tag{4.11}$$

From (4.8) and (4.11) , for request schedule $\psi_o(1)$ in both Condition 1 and Condition 2, it is shown that,

$$COST_{DWM-NR}(IA_o(1), \psi_o(1) \leq (1 + C_c + C_d) \cdot COST_{OPT^*}(IA_o^*(1), \psi_o(1))$$

The significance of proof of Part A is as follows. The Part A of Lemma 1 considers Type I request sub-schedule under two possible conditions exclusively. Observe that if there is only Type I sub-schedule in $win(o)$, then the cost for servicing ψ_o using DWM-No Replacement strategy is given by $(1 + C_c + C_d)$ competitive. This part is used when Type I sub-schedule exists in a given $win(o)$.

Proof of Part B (for Type II sub-schedule): Let $\psi_o(i) = w_o^{p_0} r_o^{p_1} r_o^{p_2} ... r_o^{p_n} (i \geq 1)$, in which $1 \leq n \leq m$ and m is the number of processors in the distributed system. Referring to the

notations in Table 2.1, there are $n_p(i)$ different processors issuing read requests, which do not belong to the allocation scheme A_o after the first write request $w_o^{p_0}$ and form the processors set $N_p(i)$.

We first find the cost of servicing the first write request $w_o^{p_0}$. Under the assumption that the processors' local database buffer has enough available space, the cost of servicing the write requests is given by,

$$COST_{DWM-NR}(w_o^{p_0}) \leq t + (t-1) \cdot C_d + C_c \tag{4.12}$$

The I/O cost of the first write request in subschedule $\psi_o(i)$ (the ith write in $win(o)$) is t, which is to save the replicas of object o into the local database of t processors. The object transmission cost at the write is $(t-1) \cdot C_d$ since the processor which issues this write needs to transmit the new version of the object to other $(t-1)$ processors to reform a new allocation scheme. Additionally, CCU needs to send control-message to invalidate the obsolete copies of the object o.

There are two cases which must be considered explicitly. The first case is that if $p_0 \notin IA_o(i-1)$ ($IA_o(i-1)$ is the allocation scheme of $\psi_o(i)$ before the first write $w_o^{p_0}$ in $\psi_o(i)$ is serviced and includes at least $F \bigcup \{p\}$), then control-messages are sent to the floating processor p in $IA_o(i-1)$. The control messages are also sent to all other processors in $N_p(i-1)/p_0$ if DWM has considers the first read requests from processors in $N_p(i-1)$ as saving read requests while servicing $\psi_o(i-1)$. Since the cost of sending a control-message to the processors in $N_p(i-1)/p_0$ is accounted in servicing the read requests in $\psi_o(i-1)$, the control-message cost of the write is simply one C_c in this equation. The second case is that when $p_0 \in IA_o(i-1)$. In this case, invalidating control messages have to be sent to all the processors in processor set $N_p(i-1)$ if DWM has considered the first read requests from processors in $N_p(i-1)$ as

saving read requests. Similarly, since the cost of sending an control-message to the processors in $N_p(i-1)$ is accounted in servicing the reads in $\psi_o(i-1)$, the "invalidating" message cost at the write is simply zero. Thus, the control-message cost at this write is at most C_c. Further, if DWM does not consider the first read requests from processors in $N_p(i-1)$ as saving read requests, then we notice that the cost to invalidate the obsolete copies of the object is at most one C_c.

Next we consider the assumption that the processor's local database buffer is full. DWM-No Replacement does not save this new version of object o in p_0's local database. Instead, it transmits new object copy to t processors. Thus, we have,

$$COST_{DWM-NR}(w_o^{p_0}) \leq t + t \cdot C_d \tag{4.13}$$

Under this assumption, the write incurs a cost of t unit for saving the object in the local database and the object transmitting cost is $t \cdot C_d$. There is no "invalidating" message cost needed, since the possible cost of sending an "invalidating" message to all processors in $N_p(i-1)$ is accounted while servicing the read requests in $\psi_o(i-1)$.

Therefore, due to an uncertainty of buffer space availability, the servicing cost of this write $w_o^{p_0}$ is given by,

$$COST_{DWM-NR}(w_o^{p_0}) \leq t + t \cdot C_d + C_c \tag{4.14}$$

The rest of the requests in $\psi_o(i)$ are all read requests. As we did in the proof of Part A of Lemma 1, we consider two conditions as follows.

Condition 1: Referring to (3.2), DWM-No Replacement may consider the first read requests issued by processors in $N_p(i)$ as saving-read requests, if $\Delta = \Big[n_r(i) - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k) - n_p(i)\Big] \cdot (C_c + C_d) > n_p(i) + n_p(i) \cdot C_c$ is satisfied.

The cost of servicing these read requests is similar to the analysis with Condition 1 of Part A. Under the assumption that the processor's local database buffer has enough available space, the cost of servicing the read requests is given by,

$$\sum_{l=1}^{n} COST_{DWM-NR}(r_o^{p_l}) = n_p(i) \cdot (1 + C_c + C_d) + n_r(i) + n_p(i) \cdot C_c \tag{4.15}$$

Next we consider the assumption if all processors' local database buffer is full. Consider the worst scenario wherein all the saving read requests are not able to save the copy of the object in its local database since the buffer has no available space. In this case, all the read requests issued by different processors which are not in the initial allocation scheme are serviced remotely. Thus, we have,

$$\begin{aligned} \sum_{l=1}^{n} COST_{DWM-NR}(r_o^{p_l}) &= \sum_{k \in N_a(i)/N_p(i)} n_r(i,k) + \Big[n_r(i) - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k)\Big] \cdot (1 + C_c + C_d) \\ &= n_r(i) + \Big[n_r(i) - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k)\Big] \cdot (C_c + C_d) \end{aligned} \tag{4.16}$$

Now, we examine each item on the right hand side of Equation (4.15) and (4.16). Referring to Condition 1, $\Delta = \Big[n_r(i) - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k) - n_p(i)\Big] \cdot (C_c + C_d) > n_p(i) + n_p(i) \cdot C_c$, we get,

$$\Big[n_r(i) - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k)\Big] \cdot (C_c + C_d) > n_p(i) \cdot (1 + 2 \cdot C_c + C_d) \tag{4.17}$$

Therefore, the upper bound of the cost in servicing the read requests in sub-schedule $\psi_o(i)$ due to an uncertainty of buffer space availability is given by,

$$\begin{aligned}\sum_{l=1}^{n} COST_{DWM-NR}(r_o^{p_l}) &\leq n_r(i) + \Big[n_r(i) \\ &\quad - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k)\Big] \cdot (C_c + C_d) \end{aligned} \tag{4.18}$$

From (4.14) and (4.18), the total cost for servicing the requests in $\psi_o(i)$ using DWM-No Replacement is given by,

$$\begin{aligned} COST_{DWM-NR}(IA_o(i), \psi_o(i)) &= COST_{DWM-NR}(w_o^{p_0}) \\ &\quad + \sum_{l=1}^{n} COST_{DWM-NR}(r_o^{p_l}) \\ &\leq n_r(i) + \Big[n_r(i) - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k)\Big] \cdot (C_c + C_d) \\ &\quad + t + t \cdot C_d + C_c \end{aligned} \tag{4.19}$$

For an optimal algorithm OPT* (as used in Part A) that satisfies t-availability constraint, every read request costs at least one unit for I/O operation. The first write request costs at least t for I/O since at least t processors need to save the new version of the object into their local database. Thus, the lower bound of servicing cost of the optimal algorithm for $\psi_o(i)$ is,

$$COST_{OPT^*}(IA_o^*(i), \psi_o(i)) \geq t + n_r(i) \tag{4.20}$$

Thus, from (4.19), we observe,

$$COST_{DWM-NR}(IA_o(i), \psi_o(i)) \leq n_r(i) + \Big[n_r(i)$$

$$
\begin{aligned}
& - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k) \Big] \cdot (C_c + C_d) \\
& + t + t \cdot C_d + C_c \\
\leq \; & n_r(i) \cdot (1 + C_c + C_d) + t \cdot (1 + C_c + C_d) \\
= \; & \Big[n_r(i) + t \Big] \cdot (1 + C_c + C_d)
\end{aligned}
\tag{4.21}
$$

Therefore, from (4.20) we obtain,

$$
COST_{DWM-NR}(IA_o(i), \psi_o(i)) \leq (1 + C_c + C_d) \cdot COST_{OPT^*}(IA^*_o(i), \psi_o(i)) \tag{4.22}
$$

Condition 1 of Part B shows the operation of DWM-No Replacement strategy to deal with Type II sub-schedule $\psi_o(i)$ when $\Delta > \Big[n_p(i) + n_p(i) \cdot C_c \Big]$.

Condition 2: Referring to (3.2), if $\Delta = \Big[n_r(i) - \sum_{k \in N_a(i)/N_p(i)} n_r(1,k) - n_p(i) \Big] \cdot (C_c + C_d) \leq n_p(i) + n_p(i) \cdot C_c$, then DWM does not consider the first read requests from different processors in $N_p(i)$ as saving read requests. In this case, processors will not save copy of the object in their local database. The total cost of servicing all the read requests in $\psi_o(i)$ is given by,

$$
\begin{aligned}
\sum_{l=1}^{n} COST_{DWM-NR}(r_o^{p_l}) &= \sum_{k \in N_a(i)/N_p(i)} n_r(i,k) + \Big[n_r(i) - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k) \Big] \cdot (1 + C_c + C_d) \\
&= n_r(i) + \Big[n_r(i) - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k) \Big] \cdot (C_c + C_d)
\end{aligned}
\tag{4.23}
$$

All the read requests issued by the processors in $N_p(i)$ cost $(1 + C_c + C_d)$ and other read requests cost one for I/O operation. The cost of servicing the write request $w_o^{p_0}$ is given by (4.14). Then, we obtain the cost for servicing $\psi_o(i)$ by,

$$\begin{aligned} COST_{DWM-NR}(IA_o(i), \psi_o(i)) &= COST_{DWM-NR}(w_o^{p_0}) \\ &\quad + \sum_{l=1}^{n} COST_{DWM-NR}(r_o^{p_l}) \\ &\leq n_r(i) + \Big[n_r(i) - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k)\Big] \cdot (C_c + C_d) \\ &\quad + t + t \cdot C_d + C_c \end{aligned} \tag{4.24}$$

The lower bound of servicing such a request schedule $\psi_o(i)$ is the same as (4.20) in Condition 1, given by $COST_{OPT^*}(IA_o^*(i), \psi_o(i)) \geq t + n_r(i)$. Then, by comparing (4.24) and (4.20), we conclude,

$$\begin{aligned} COST_{DWM-NR}(IA_o(i), \psi_o(i)) &\leq n_r(i) + \Big[n_r(i) \\ &\quad - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k)\Big] \cdot (C_c + C_d) \\ &\quad + t + t \cdot C_d + C_c \\ &\leq n_r(i) \cdot (1 + C_c + C_d) + t \cdot (1 + C_c + C_d) \\ &= \Big[n_r(i) + t\Big] \cdot (1 + C_c + C_d) \end{aligned} \tag{4.25}$$

Therefore, we obtain,

$$COST_{DWM-NR}(IA_o(i), \psi_o(i) \leq (1 + C_c + C_d) \cdot COST_{OPT^*}(IA_o^*(i), \psi_o(i)) \tag{4.26}$$

From (4.22) and (4.26) , for request schedule $\psi_o(i)$ in both Condition 1 and Condition 2, it is shown that,

$$COST_{DWM-NR}(IA_o(i), \psi_o(i) \leq (1 + C_c + C_d) \cdot COST_{OPT^*}(IA_o^*(i), \psi_o(i)) \tag{4.27}$$

Thus, the result of Part B is immediate.

The usefulness of proof of Part B of Lemma 1 is identical to Part A, except the fact that it considers Type II sub-schedule exclusively. Thus, Lemma 1 follows from the proof of Part A and Part B for any type of request schedule $\psi_o(i)$. □

We use the results in Part A and Part B of Lemma 1 to prove the Theorem 1.

Proof of Theorem 1: The total cost for servicing the request schedule window $win(o)$ by DWM-No Replacement is given by,

$$COST_{DWM-NR}(IA_o, \psi_o) = \sum_{i=1}^{n} COST_{DWM-NR}(IA_o(i), \psi_o(i)) \tag{4.28}$$

$IA_o(i)$ is the initial allocation scheme of object o on $\psi_o(i)$, which is formed after request schedule $\psi_o(i-1)$ is serviced by DWM-No Replacement.

Similarly, the service cost of the same request schedules in $win(o)$ by an optimal offline algorithm is given by,

$$COST_{OPT^*}(IA_o, \psi_o) = \sum_{i=1}^{n} COST_{OPT^*}(IA_o^*(i), \psi_o(i)) \tag{4.29}$$

$IA_o^*(i)$ is the initial allocation scheme of object o on $\psi_o(i)$, which is formed after servicing $\psi_o(i-1)$ using the optimal algorithm. Note that $IA_o(1) = IA_o^*(1) = IA_o$.

From Part A and Part B of Lemma 1, we know that for any $\psi_o(i)$ in $win(o)$,

$$COST_{DWM-NR}(IA_o(i), \psi_o(i)) \leq (1 + C_c + C_d) \cdot COST_{OPT^*}(IA_o^*(i), \psi_o(i)) \tag{4.30}$$

Therefore, from (4.28), (4.29) and (4.30), we conclude,

$$\begin{aligned} COST_{DWM-NR}(IA_o, \psi_o) &= \sum_{i=1}^{n} COST_{DWM-NR}(IA_o(i), \psi_o(i)) \\ &\leq (1 + C_c + C_d) \cdot \sum_{i=1}^{n} COST_{OPT^*}(IA_o^*(i), \psi_o(i)) \\ &\leq (1 + C_c + C_d) \cdot COST_{OPT^*}(IA_o^*, \psi_o) \end{aligned} \tag{4.31}$$

Hence the proof. □

4.2.2 Competitive ratio of dynamic allocation DWM-Replacement

In this section we present the competitive ratio of DWM-Replacement algorithm. The cost functions of DWM-Replacement strategy were presented by (3.4) and (3.6) in the previous chapter. In terms of our model, DWM-Replacement considers a $win(o)$ of an object o in an arbitrary request schedule ψ. Each request is serviced according to DWM as a non-saving read, a saving read or a write. During the servicing process, when the processor's local database is full, object replacement Strategy II (LRU) and III (LFU) pick and purge one object in use in the local database to give space to store the newly incoming copy of the object. Note that DWM selects a set of processors F of size $(t-1)$ and a processor $p \notin F$, such that $F \bigcup \{p\}$ is the initial allocation scheme of size t.

Theorem 2 *For any integer $t \geq 1$, DWM-Replacement algorithm is $(2 + 3 \cdot \frac{C_c}{C_d})$-competitive.*

In the proof of Theorem 2, we follow the same steps in the proof of Theorem 1. We prove the following lemma before we realizing the result of Theorem 2.

Lemma 2 *Suppose that $IA_o(i)$ is the initial allocation scheme of object o on $\psi_o(i)$ and $\psi_o(i) = \{\psi_o^{ss1}, \psi_o^{ss2}\}$ in $win(o)$. Suppose that $IA_o^*(i)$ is the initial allocation scheme of object o on $\psi_o(i)$*

using an optimal algorithm that satisfies t-availability constraint. Then,

$$COST_{DWM-R}(IA_o(i), \psi_o(i)) \leq (2 + 3 \cdot \frac{C_c}{C_d}) \cdot COST_{OPT^*}(IA^*_o(i), \psi_o(i))$$

Proof. Similar to the proof of Lemma 1, we split the $win(o)$ into several subschedules, each one of which belongs to one of the two special types of sub-schedules, Type I sub-schedule ψ_o^{ss1} and Type II sub-schedule ψ_o^{ss2}. Type I sub-schedule consists of read requests only. Type II sub-schedule consists of a write request followed by zero or more read requests. Then, we can split our proof into two parts and show that the cost of DWM-R on both types of sub-schedules is at most $(2+3\cdot\frac{C_c}{C_d})$ times as much as the cost of an optimum offline algorithm.

Proof of Part A (for read-only Type I sub-schedule): Notice that if Type I sub-schedule exists, it must be at the beginning of $win(o)$. Thus, we suppose that $\psi_o(1) = r_o^{p_1} r_o^{p_2} r_o^{p_3} ... r_o^{p_n}$ comprises read requests only, which come before the first write request. Let $n_p(1)$ be the number of different reading processors of $\psi_o(1)$ which are not in $IA_o(1)$, say $F \bigcup \{p\}$. Similarly to Part A of Lemma 1, we consider the following two conditions to analyze the cost bound comparison.

Condition 1: Referring to the working style of DWM in Section 3.1, DWM may consider all the first read requests issued by the different processors in $\psi_o(1)$ which are not in $F \bigcup \{p\}$ as saving-read requests if $\Delta = \Big[n_r(1) - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k) - n_p(1)\Big] \cdot (C_c + C_d) > n_p(1) + n_p(1) \cdot C_c$ (refer to Equation (3.1)). Under the assumption that the processor's local database buffer has enough available space, the total cost incurred by servicing $\psi_o(1)$ is given by,

$$\begin{aligned} COST_{DWM-R}(IA_o(1), \psi_o(1)) &= \sum_{l=1}^{n} COST_{DWM-R}(r_o^{p_l}) \\ &= n_p(1) \cdot (1 + C_c + C_d) + n_r(1) + n_p(1) \cdot C_c \\ &= n_r(1) + n_p(1) \cdot (1 + 2 \cdot C_c + C_d) \end{aligned} \tag{4.32}$$

Because the first read requests issued by processors in $N_p(1)$ will save the copy of object in processors' local database, thus, each read costs $(1 + C_c + C_d)$ more than all other read requests in $\psi_o(1)$. Additionally, each read request, no matter if it is serviced locally or remotely, costs one unit for outputting the object o from a local database of a processor. Besides, if DWM considers these read requests into saving-read requests, then this incurs additional more "invalidation cost" by at most $n_p(1) \cdot C_c$ than the cost when DWM does not consider these requests as saving read requests, since the future write request in next subschedule needs to invalidate all the obsolete copies of object o.

Next we consider the assumption that all processors' local database buffer is full. In this case, DWM-Replacement algorithm evicts an object in the processor's local database by either object replacement Strategy II or object replacement Strategy III. Every eviction incurs one additional control message to inform CCU of the change of the allocation scheme of the object which is evicted from the processor's local database. Suppose that the worst case is that all the saving read requests incur such object replacement behaviors at processors which issued these saving reads. Thus, extra cost of $n_p(1) \cdot C_c$ is needed compared with (4.32). Then, we have,

$$\begin{aligned} COST_{DWM-R}(IA_o(1), \psi_o(1)) &= \sum_{l=1}^{n} COST_{DWM-R}(r_o^{p_l}) \\ &= n_r(1) + n_p(1) \cdot (1 + 2 \cdot C_c + C_d) + n_p(1) \cdot C_c \\ &= n_r(1) + n_p(1) \cdot C_d + n_p(1) + 3 \cdot n_p(1) \cdot C_c \end{aligned} \quad (4.33)$$

Now, we examine each item on the right hand side of Equation (4.32) and (4.34). The upper bound of the cost in servicing the request schedule $\psi_o(1)$ due to an uncertainty of buffer space availability is given by,

$$COST_{DWM-R}(IA_o(1), \psi_o(1)) \leq n_r(1) + n_p(1) \cdot C_d + n_p(1) + 3 \cdot n_p(1) \cdot C_c \quad (4.34)$$

Next, we find the lower bound of the cost by an optimal offline algorithm OPT* without buffer constraints. Let us consider the same request schedule $\psi_o(1)$. For an optimal algorithm OPT* that satisfies t-availability constraint, there are also $n_p(1)$ processors in processor set $N_p(1)$ that have no replicas of object o in their local databases. Therefore, the object transmitting cost is at least $n_p(1) \cdot C_d$. Besides, each read request in $\psi_o(1)$ costs at least one unit for the I/O cost to output the object from the local database, hence,

$$COST_{OPT^*}(IA^*_o(1), \psi_o(1)) \geq n_p(1) \cdot C_d + n_r(1) \quad (4.35)$$

Then, from (4.34), we derive,

$$\begin{aligned} COST_{DWM-R}(IA_o(1), \psi_o(1)) &\leq n_r(1) + n_p(1) \cdot C_d + n_p(1) + 3 \cdot n_p(1) \cdot C_c \\ &\leq 2 \cdot \Big[n_r(1) + n_p(1) \cdot C_d\Big] + 3 \cdot n_p(1) \cdot C_c + 3 \cdot \frac{C_c}{C_d} \cdot n_r(1) \\ &= 2 \cdot \Big[n_r(1) + n_p(1) \cdot C_d\Big] + 3 \cdot \frac{C_c}{C_d} \cdot \Big[n_r(1) + n_p(1) \cdot C_d\Big] \end{aligned} \quad (4.36)$$

Therefore, by comparing (4.36) and (4.35), we conclude,

$$COST_{DWM-R}(IA_o(1), \psi_o(1)) \leq (2 + 3 \cdot \frac{C_c}{C_d}) \cdot COST_{OPT^*}(IA^*_o(1), \psi_o(1)) \quad (4.37)$$

Condition 2: Referring to Equation (3.1), if $\Delta = \Big[n_r(1) - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k) - n_p(1)\Big] \cdot (C_c + C_d) \leq n_p(1) + n_p(1) \cdot C_c$, then DWM-Replacement should not consider the first read requests from different processors in $N_p(1)$ as saving read requests. In this case, processors will not save copies of objects in their local databases. Then, we obtain the cost for servicing $\psi_o(1)$ by,

$$\begin{aligned} COST_{DWM-R}(IA_o(1), \psi_o(1)) &= \sum_{l=1}^{n} COST_{DWM-R}(r_o^{p_l}) \\ &= \sum_{k \in N_a(1)/N_p(1)} n_r(1,k) + \Big[n_r(1) \\ &\quad - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k)\Big] \cdot (1 + C_c + C_d) \\ &= n_r(1) + \Big[n_r(1) \\ &\quad - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k)\Big] \cdot (C_c + C_d) \end{aligned} \tag{4.38}$$

All the read requests issued by the processors in $N_p(1)$ are serviced remotely, thus every such read costs $(1 + C_c + C_d)$. All other read requests are serviced locally, incurring one unit cost for every I/O operation.

From Condition 2, $\Delta = \Big[n_r(1) - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k) - n_p(1)\Big] \cdot (C_c + C_d) \leq n_p(1) + n_p(1) \cdot C_c$, we obtain,

$$\Big[n_r(1) - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k)\Big] \cdot (C_c + C_d) \leq n_p(1) \cdot (1 + 2 \cdot C_c + C_d) \tag{4.39}$$

As in Condition 1 of Part A, we compare (4.38) with a lower cost bound by an optimal offline algorithm OPT* which satisfies t-availability constraint (given by (4.35)) and consider Condition 2 (4.39). Then, we can derive,

$$\begin{aligned} COST_{DWM-R}(IA_o(1), \psi_o(1)) &= n_r(1) + \Big[n_r(1) \\ &\quad - \sum_{k \in N_a(1)/N_p(1)} n_r(1,k)\Big] \cdot (C_c + C_d) \\ &\leq n_r(1) + n_p(1) \cdot C_d + n_p(1) + 2 \cdot n_p(1) \cdot C_c \\ &\leq 2 \cdot \Big[n_r(1) + n_p(1) \cdot C_d\Big] + 3 \cdot \frac{C_c}{C_d} \cdot \Big[n_r(1) + n_p(1) \cdot C_d\Big] \end{aligned}$$

Therefor, we conclude,

$$COST_{DWM-R}(IA_o(1), \psi_o(1)) \leq (2 + 3 \cdot \frac{C_c}{C_d}) \cdot COST_{OPT^*}(IA_o^*(1), \psi_o(1)) \tag{4.40}$$

From (4.37) and (4.40), the Part A of Lemma 2 considers Type I request sub-schedule under two possible conditions exclusively. Observe that if there is only Type I sub-schedule in $win(o)$, then the cost for servicing ψ_o using DWM-Replacement strategy is given by $(2+3\cdot\frac{C_c}{C_d})$-competitive. This part is used when Type I sub-schedule exists in a given $win(o)$.

Proof of Part B (for Type II sub-schedule): Let $\psi_o(i) = w_o^{p_0} r_o^{p_1} r_o^{p_2} ... r_o^{p_n}$, $1 \leq n \leq m$ and m is the number of processors in the distributed system. There are $n_p(i)$ different processors issuing read requests. These processors do not belong to the allocation scheme A_o after the first write request $w_o^{p_0}$ and form the processors set $N_p(i)$.

We first find the cost of servicing the first write request $w_o^{p_0}$. Under the assumption that the processors' local database buffer has enough available space, the cost of servicing the write requests is given by,

$$COST_{DWM-R}(w_o^{p_0}) \leq t + (t-1) \cdot C_d + C_c \tag{4.41}$$

A write request incurs $(t-1) \cdot C_d$ cost due to data-message cost and t for I/O cost, since

the new version copy of the object o need to be sent to $(t-1)$ other processors and to be saved in local databases of t processors. Additionally, as we analyzed in proof of Part B of Lemma 1 in Section 4.2.1, control-messages must be sent to other processors to invalidate the redundant replicas of object o. Since the cost of sending invalidating messages to all processors (at most) in $N_p(i-1)$ is already accounted while servicing the read requests in $\psi_o(i-1)$ if these reads are considered as saving read requests, we only consider if the copy of object o at floating processor p need to be invalidated. If $p_0 \notin IA_o(i-1)$ ($IA_o(i-1)$ is the allocation scheme of $\psi_o(i)$ before the first write $w_o^{p_0}$ in $\psi_o(i)$ is serviced and includes at least $F \bigcup \{p\}$), a control message need to be sent to the floating processor p in order to invalidate the obsolete copy of the object. If $p_0 \in IA_o(i-1)$, control message cost is simply zero. Thus, here we have at most one C_c for the invalidation of the object copy at this write request. Further, if DWM-Replacement does not consider the read requests as saving reads, the cost to invalidate the redundant copies of the object is at most one C_c.

Next we consider the assumption that the processor's local database buffer is full. DWM-Replacement evicts an object in the processor's local database by either object replacement Strategy II or object replacement Strategy III. Note that each eviction incurs one additional control message to inform CCU of the change of the allocation scheme of the object which is evicted from the processor's local database. Thus, we obtain the servicing cost of this write due to an uncertainty of buffer space availability by,

$$COST_{DWM-R}(w_o^{p_0}) \leq t + (t-1) \cdot C_d + C_c + C_c \tag{4.42}$$

The rest of the requests in $\psi_o(i)$ are all read requests. As we did in the proof of Part A of Lemma 2, we consider two conditions as follows.

Condition 1: Referring to (3.2), if $\Delta = \Big[n_r(i) - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k) - n_p(i)\Big] \cdot (C_c + C_d) > n_p(i) + n_p(i) \cdot C_c$, DWM-Replacement may consider the first read requests by processors in $N_p(i)$ as saving-read requests. We follow the same analysis as we did in Condition 1 of Part A to find the upper bound cost for servicing these read requests. First, under the assumption that the processor's local database buffer has enough available space, the total cost incurred by servicing reads in $\psi_o(i)$ is given by,

$$\sum_{l=1}^{n} COST_{DWM-R}(r_o^{p_l}) = n_r(i) + n_p(i) \cdot (1 + 2 \cdot C_c + C_d) \tag{4.43}$$

Then, we consider the assumption that all processors' local database buffer is full. In this case, DWM-Replacement algorithm evicts an object in the processor's local database by either object replacement Strategy II or object replacement Strategy III. Every eviction incurs one additional control message cost C_c. Thus, extra cost of $n_p(1) \cdot C_c$ is needed compared with (4.43). Then, we have,

$$\sum_{l=1}^{n} COST_{DWM-R}(r_o^{p_l}) = n_r(i) + n_p(i) \cdot C_d + n_p(i) + 3 \cdot n_p(i) \cdot C_c \tag{4.44}$$

By examining each item on the right hand side of Equation (4.43) and (4.44), the upper bound of the cost in servicing the read requests in sub-schedule $\psi_o(i)$ due to an uncertainty of buffer space availability is given by,

$$\sum_{l=1}^{n} COST_{DWM-R}(r_o^{p_l}) = n_r(i) + n_p(i) \cdot C_d + n_p(i) + 3 \cdot n_p(i) \cdot C_c \tag{4.45}$$

From (4.42) and (4.45), the total cost for servicing the requests in $\psi_o(i)$ using DWM-Replacement

is given by,

$$\begin{aligned} COST_{DWM-R}(IA_o(i), \psi_o(i)) &= COST_{DWM-R}(w_o^{p_0}) + \sum_{l=1}^{n} COST_{DWM-R}(r_o^{p_l}) \\ &\leq n_r(i) + n_p(i) \cdot C_d + n_p(i) + 3 \cdot n_p(i) \cdot C_c \\ &\quad + t + (t-1) \cdot C_d + 2 \cdot C_c \end{aligned} \tag{4.46}$$

Next, we find the lower bound of servicing cost by an optimal offline algorithm OPT* that satisfies t-availability constraint. There are $n_a(i)$ different reading processors in sub-schedule $\psi_o(i)$. Denote this set of reading processors in $\psi_o(i)$ by $N_a(i)$. There are two cases (Case A and Case B) that we need to consider when we try to find the lower cost bound.

Case A: $\underline{n_a(i) > t}$. Then, the first write request incurs at least $\Big[n_a(i) - 1\Big]$ data-message cost, since all processors in $N_a(i)$ need the new copy of the object and assuming that writing processor p_0 is also in $N_a(i)$. This write also costs at least t units of I/O cost. Additionally, each read request in $n_r(i)$ costs at least one unit for I/O operation. Hence, the total cost of the optimal offline algorithm for $\psi_o(i)$ is given by,

$$COST_{OPT^*}(IA_o^*(i), \psi_o(i)) \geq \Big[n_a(i) - 1\Big] \cdot C_d + t + n_r(i) \tag{4.47}$$

Then, from (4.46), we derive,

$$\begin{aligned} COST_{DWM-R}(IA_o(i), \psi_o(i)) &\leq n_r(i) + n_p(i) \cdot C_d + n_p(i) + 3 \cdot n_p(i) \cdot C_c \\ &\quad + t + (t-1) \cdot C_d + 2 \cdot C_c \\ &\leq n_r(i) + t + \Big[(n_a(i) - t) + (t-1)\Big] \cdot C_d \end{aligned}$$

$$
\begin{aligned}
& + \Big[n_p(i) - (n_a(i) - t)\Big] \cdot C_d + n_r(i) + t - t \\
& + 3 \cdot \frac{C_c}{C_d} \cdot n_r(i) + 3 \cdot n_p(i) \cdot C_c \\
& + 3 \cdot \frac{C_c}{C_d} \cdot t - 3 \cdot \frac{C_c}{C_d} \cdot t + 2 \cdot C_c \\
\leq\ & 2 \cdot \Big(n_r(i) + t + \Big[(n_a(i) - 1)\Big] \cdot C_d\Big) \\
& + 3 \cdot \frac{C_c}{C_d} \cdot \Big(n_r(i) + t + \Big[n_a(i) - 1\Big] \cdot C_d\Big) \\
& + 5 \cdot C_c - 3 \cdot \frac{C_c}{C_d} \cdot t - t \\
& (\text{since } n_a(i) > t, n_a(i) > n_p(i) \text{ and } C_d \geq C_c)
\end{aligned}
$$

Therefore, from (4.47) we conclude,

$$COST_{DWM-R}(IA_o(i), \psi_o(i)) \leq (2 + 3 \cdot \frac{C_c}{C_d}) \cdot COST_{OPT^*}(IA_o^*(i), \psi_o(i)) + K \qquad (4.48)$$

where $K = (5 \cdot C_c - 3 \cdot \frac{C_c}{C_d} \cdot t - t)$, is a constant.

Case B: $\underline{n_a(i) \leq t}$. Then, the I/O cost of the first write is t. This write also incurs $(t-1) \cdot C_d$ data-message cost since it will send the copy of object o to all the processors in $N_a(i)$. Each read costs at least one I/O cost. Hence,

$$COST_{OPT^*}(IA_o^*(i), \psi_o(i)) \geq (t-1) \cdot C_d + t + n_r(i) \qquad (4.49)$$

Then, from (4.46), we can see,

$$
\begin{aligned}
COST_{DWM-R}(IA_o(i), \psi_o(i)) \leq\ & n_r(i) + n_p(i) \cdot C_d + n_p(i) + 3 \cdot n_p(i) \cdot C_c \\
& + t + (t-1) \cdot C_d + 2 \cdot C_c \\
\leq\ & n_r(i) + t + (t-1) \cdot C_d + \Big[n_p(i) - 1\Big] \cdot C_d + t + n_r(i) \\
& + C_d - t + 3 \cdot \Big[n_p(i) - 1\Big] \cdot C_c + 3 \cdot C_c + 3 \cdot \frac{C_c}{C_d} \cdot n_r(i)
\end{aligned}
$$

$$
\begin{aligned}
& +3 \cdot \frac{C_c}{C_d} \cdot t - 3 \cdot \frac{C_c}{C_d} \cdot t + 2 \cdot C_c \\
\leq\ & 2 \cdot \Big[n_r(i) + t + (t-1) \cdot C_d \Big] \\
& +3 \cdot \frac{C_c}{C_d} \cdot \Big[n_r(i) + t + (t-1) \cdot C_d \Big] \\
& +C_d + 5 \cdot C_c - 3 \cdot \frac{C_c}{C_d} \cdot t - t \\
& (\text{since } n_p(i) \leq n_a(i) \leq t \text{ and } C_d \geq C_c)
\end{aligned}
$$

Therefore, from (4.49) we conclude,

$$COST_{DWM-R}(IA_o(i), \psi_o(i)) \leq (2 + 3 \cdot \frac{C_c}{C_d}) \cdot COST_{OPT^*}(IA^*_o(i), \psi_o(i)) + K \qquad (4.50)$$

where $K = (C_d + 5 \cdot C_c - 3 \cdot \frac{C_c}{C_d} \cdot t - t)$, is a constant.

Condition 1 of Part B in Lemma 2 shows the cost bounds comparison of DWM-Replacement algorithm in Type II request sub-schedule $\psi_o(i)$ when $\Delta > \Big[n_p(i) + n_p(i) \cdot C_c \Big]$.

Condition 2: Referring to (3.2), if $\Delta = \Big[n_r(i) - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k) - n_p(i) \Big] \cdot (C_c + C_d) \leq n_p(i) + n_p(i) \cdot C_c$, DWM-Replacement will not consider the first read requests from each of processors in $N_p(i)$ as saving-read requests. Then, the total cost incurred by servicing all reads in $\psi_o(i)$ is given by,

$$\sum_{l=1}^{n} COST_{DWM-R}(r_o^{p_l}) = n_r(i) + \Big[n_r(i) - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k) \Big] \cdot (C_c + C_d) \qquad (4.51)$$

All the read requests issued by processors in $N_p(i)$ incur $(1 + C_c + C_d)$ units of the cost for each of them and all other read requests incur only one unit for each of them for I/O operation. The cost of servicing the first write request $w_o^{p_0}$ is already given by (4.42). Thus, the total cost for servicing the requests in $\psi_o(i)$ using DWM-Replacement is given by,

$$COST_{DWM-R}(IA_o(i), \psi_o(i)) \ = \ COST_{DWM-R}(w_o^{p_0}) + \sum_{l=1}^{n} COST_{DWM-R}(r_o^{p_l})$$

$$\begin{aligned} &\leq n_r(i) + \Big[n_r(i) - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k)\Big] \cdot (C_c + C_d) \\ &\quad + t + (t-1) \cdot C_d + 2 \cdot C_c \end{aligned} \tag{4.52}$$

Given the Condition 2, $\Big[n_r(i) - \sum_{k \in N_a(i)/N_p(i)} n_r(1,k) - n_p(i)\Big] \cdot (C_c + C_d) \leq n_p(i) + n_p(i) \cdot C_c$, we have,

$$\Big[n_r(i) - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k)\Big] \cdot (C_c + C_d) \leq n_p(i) \cdot (1 + 2 \cdot C_c + C_d) \tag{4.53}$$

Next, when we find the lower bound of servicing cost by an optimal offline algorithm OPT* that satisfies t-availability constraint, we also consider two cases as in Condition 1 of Part B.

Case A: $\underline{n_a(i) > t}$. The total cost for servicing $\psi_o(i)$ using an optimal offline algorithm is identical to (4.47). Thus, from (4.52), we can derive,

$$\begin{aligned} COST_{DWM-R}(IA_o(i), \psi_o(i)) \;&\leq\; n_r(i) + \Big[n_r(i) - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k)\Big] \cdot (C_c + C_d) \\ &\quad + t + (t-1) \cdot C_d + 2 \cdot C_c \end{aligned}$$

by using Condition 2 (4.53),

$$\begin{aligned} COST_{DWM-R}(IA_o(i), \psi_o(i)) \;&\leq\; n_r(i) + n_p(i) \cdot (1 + 2 \cdot C_c + C_d) \\ &\quad + t + (t-1) \cdot C_d + 2 \cdot C_c \\ &\leq\; n_r(i) + t + \big[(n_a(i) - t) + (t-1)\big] \cdot C_d \\ &\quad + \Big[n_p(i) - (n_a(i) - t)\Big] \cdot C_d + n_r(i) + t - t \\ &\quad + 3 \cdot \frac{C_c}{C_d} \cdot n_r(i) + 3 \cdot n_p(i) \cdot C_c \\ &\quad + 3 \cdot \frac{C_c}{C_d} \cdot t - 3 \cdot \frac{C_c}{C_d} \cdot t + 2 \cdot C_c \end{aligned}$$

$$
\begin{aligned}
&\leq 2 \cdot \Big(n_r(i) + t + \Big[(n_a(i) - 1)\Big] \cdot C_d\Big) \\
&\quad +3 \cdot \frac{C_c}{C_d} \cdot \Big(n_r(i) + t + \Big[(n_a(i) - 1)\Big] \cdot C_d\Big) \\
&\quad +5 \cdot C_c - 3 \cdot \frac{C_c}{C_d} \cdot t - t \\
&\quad (\text{since } n_a(i) > t, n_a(i) > n_p(i) \text{ and } C_d \geq C_c)
\end{aligned}
$$

Therefore, from (4.47) we conclude,

$$
COST_{DWM-R}(IA_o(i), \psi_o(i)) \leq (2 + 3 \cdot \frac{C_c}{C_d}) \cdot COST_{OPT^*}(IA^*_o(i), \psi_o(i)) + K \qquad (4.54)
$$

where $K = (5 \cdot C_c - 3 \cdot \frac{C_c}{C_d} \cdot t - t)$, is a constant.

Case B: $\underline{n_a(i) \leq t}$. The total cost for servicing $\psi_o(i)$ using an optimal offline algorithm is identical to (4.49). Thus, from (4.52), we can derive,

$$
\begin{aligned}
COST_{DWM-R}(IA_o(i), \psi_o(i)) \leq\ & n_r(i) + \Big[n_r(i) - \sum_{k \in N_a(i)/N_p(i)} n_r(i,k)\Big] \cdot (C_c + C_d) \\
& +t + (t-1) \cdot C_d + 2 \cdot C_c
\end{aligned}
$$

by using Condition 2 (4.53),

$$
\begin{aligned}
COST_{DWM-R}(IA_o(i), \psi_o(i)) &\leq n_r(i) + n_p(i) \cdot (1 + 2 \cdot C_c + C_d) \\
&\quad +t + (t-1) \cdot C_d + 2 \cdot C_c \\
&\leq n_r(i) + t + (t-1) \cdot C_d + \Big[n_p(i) - 1\Big] \cdot C_d + t + n_r(i) \\
&\quad +C_d - t + 3 \cdot \Big[n_p(i) - 1\Big] \cdot C_c + 3 \cdot C_c + 3 \cdot \frac{C_c}{C_d} \cdot n_r(i) \\
&\quad +3 \cdot \frac{C_c}{C_d} \cdot t - 3 \cdot \frac{C_c}{C_d} \cdot t + 2 \cdot C_c \\
&\leq 2 \cdot \Big[n_r(i) + t + (t-1) \cdot C_d\Big] \\
&\quad +3 \cdot \frac{C_c}{C_d} \cdot \Big[n_r(i) + t + (t-1) \cdot C_d\Big] \\
&\quad +C_d + 5 \cdot C_c - 3 \cdot \frac{C_c}{C_d} \cdot t - t \\
&\quad (\text{since } n_p(i) \leq n_a(i) \leq t \text{ and } C_d \geq C_c)
\end{aligned}
$$

Therefore, from (4.49) we conclude,

$$COST_{DWM-R}(IA_o(i), \psi_o(i)) \leq (2 + 3 \cdot \frac{C_c}{C_d}) \cdot COST_{OPT^*}(IA^*_o(i), \psi_o(i)) + K \qquad (4.55)$$

where $K = (C_d + 5 \cdot C_c - 3 \cdot \frac{C_c}{C_d} \cdot t - t)$, is a constant.

From (4.48), (4.50), (4.54) and (4.55), the result of Part B of Lemma 2 is immediate. Part B of Lemma 2 considers Type II sub-schedules exclusively. Thus, Lemma 2 follows from the proof of Part A and Part B for any type of request schedule $\psi_o(i)$. □

We use the results in Part A and Part B of Lemma 2 to prove the Theorem 2.

Proof of Theorem 2: The total cost for servicing the request schedule window $win(o)$ by DWM-Replacement is given by,

$$COST_{DWM-R}(IA_o, \psi_o) = \sum_{i=1}^{n} COST_{DWM-R}(IA_o(i), \psi_o(i)) \qquad (4.56)$$

$IA_o(i)$ is the initial allocation scheme of object o on $\psi_o(i)$, which is formed after request schedule $\psi_o(i-1)$ is serviced by DWM-Replacement.

Similarly, the service cost of the same request schedules in $win(o)$ by an optimal offline algorithm is given by,

$$COST_{OPT^*}(IA_o, \psi_o) = \sum_{i=1}^{n} COST_{OPT^*}(IA^*_o(i), \psi_o(i)) \qquad (4.57)$$

$IA^*_o(i)$ is the initial allocation scheme of object o on $\psi_o(i)$, which is formed after servicing $\psi_o(i-1)$ using the optimal algorithm. Note that $IA_o(1) = IA^*_o(1) = IA_o$.

From Lemma 2, we know that for any type of $\psi_o(i)$ in $win(o)$,

$$COST_{DWM-R}(IA_o(i), \psi_o(i)) \leq (2 + 3 \cdot \frac{C_c}{C_d}) \cdot COST_{OPT^*}(IA_o^*(i), \psi_o(i)) + K \qquad (4.58)$$

Therefore, from (4.56), (4.57) and (4.58), we conclude,

$$\begin{aligned} COST_{DWM-R}(IA_o, \psi_o) &= \sum_{i=1}^{n} COST_{DWM-R}(IA_o(i), \psi_o(i)) \\ &\leq (2 + 3 \cdot \frac{C_c}{C_d}) \cdot \sum_{i=1}^{n} COST_{OPT^*}(IA_o^*(i), \psi_o(i)) + K \\ &\leq (2 + 3 \cdot \frac{C_c}{C_d}) \cdot COST_{OPT^*}(IA_o^*, \psi_o) + K \qquad (4.59) \end{aligned}$$

Hence the proof. □

4.2.3 Cost comparison analysis

The cost-comparison results, discussed in this section, are obtained for the DWM-No Replacement and DWM-Replacement algorithms operating in the normal environment Model A, which is in the absence of failures. For simplicity, we have normalized the cost by taking $C_{io} = 1$. Thus, C_c is the ratio of a control message cost to the I/O cost of a read-write request. Similarly, C_d is the ratio of a data message cost to the C_{io}.

In Model A, we obtain the following results. We show that the DWM-No Replacement algorithm is $(1 + C_c + C_d)$-competitive. Then we show that the DWM-Replacement algorithm is $(2 + 3 \cdot \frac{C_c}{C_d})$-competitive in general. Since a data message can not be less costly than a control message, we have $C_d \geq C_c$. Therefore, $(2 + 3 \cdot \frac{C_c}{C_d}) \leq 5$. Then we can find, when $C_d > (4 - C_c)$, the DWM-Replacement algorithm is superior to the DWM-No Replacement algorithm. Since in this case, $(1 + C_c + C_d) > 5$.

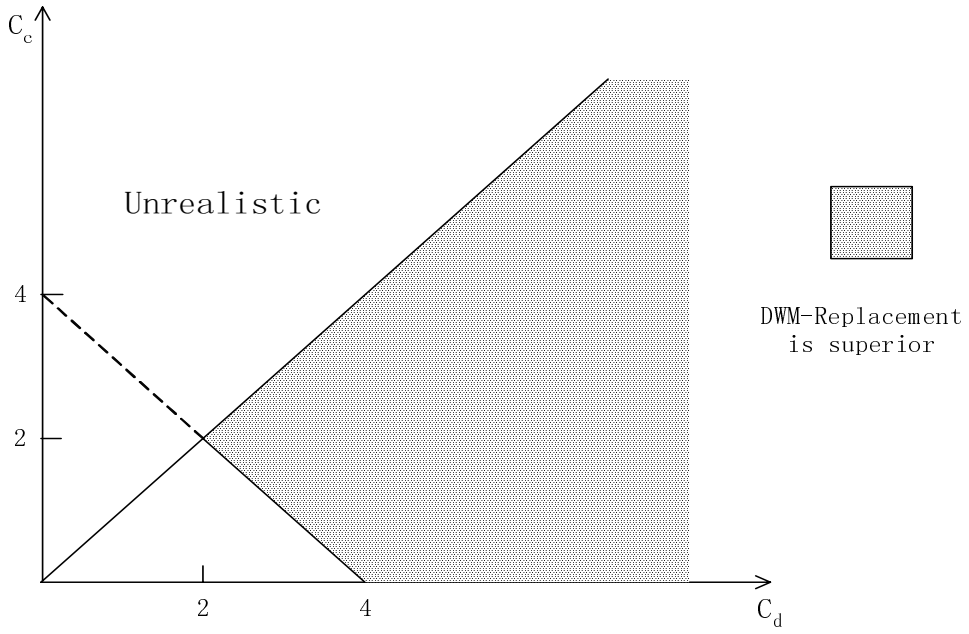


Figure 4.1: Superiority of DWM-Replacement

The results of the comparison are summarized in Fig. 4.1. The figure indicates the area on the C_c and C_d plane for which the DWM-Replacement algorithm is strictly superior. The area in which $C_c > C_d$ is marked "Not true", since a data message includes the object-id and operation field (read, write, or invalidate) fields, as well as the copy of object content, while the control message includes the object-id and operation (read, write, or invalidate) fields only.

When we assume $C_d \gg C_c$, then $(2+3 \cdot \frac{C_c}{C_d}) \leq 2$. In this case, when $C_d > (1-C_c)$, the DWM-Replacement is superior to the DWM-No Replacement algorithm, since $(1 + C_c + C_d) > 2$. The results of the comparison are summarized in Fig. 4.2. The figure shows the area for which the DWM-Replacement algorithm is strictly superior as in Fig. 4.1 and the area for which DWM-No Replacement algorithm is strictly superior. The area marked "Unknown" represents the C_c and C_d values for which the superiority of either DWM-No Replacement or DWM-Replacement is currently uncertain. The reason for this uncertainty is that the uncertain ratio of C_c/C_d is the key factor to know whether the DWM-Replacement algorithm

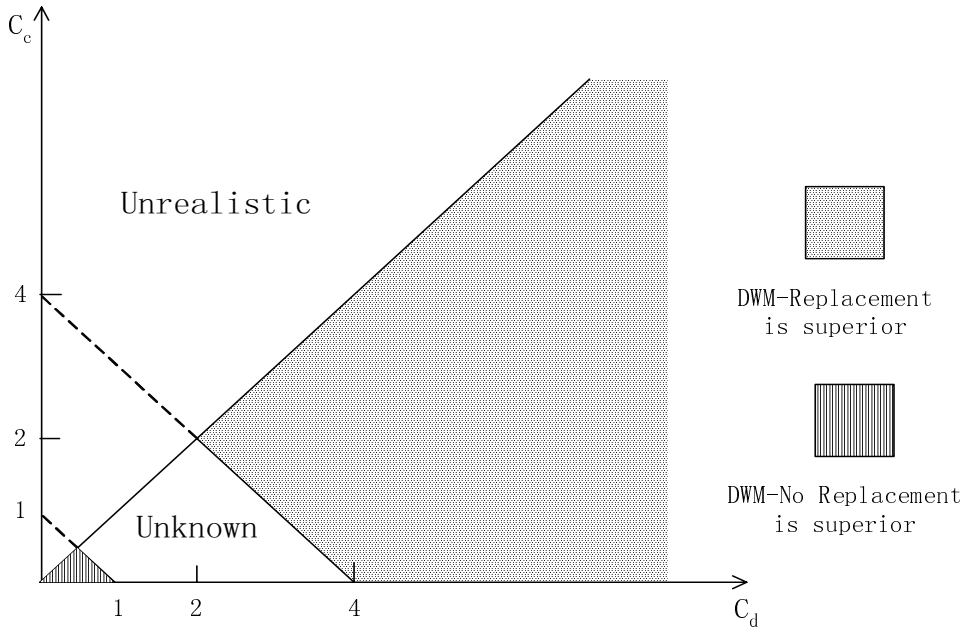


Figure 4.2: Superiority of DWM-Replacement and DWM-No Replacement

is superior to the DWM-No Replacement algorithm or vice versa when the values of C_c and C_d fall in this area.

4.3 Competitive Ratios of Different Strategies In Model B

In this section, we will prove that our algorithms are still competitive and present the competitiveness factors of dynamic data allocation and replication algorithms with buffer constraints in the Heterogeneous Object Sizes Model (Model B). In Model B, different objects have different sizes. The cost functions of different algorithms for servicing the read-write requests have been introduced in Section 3.3. Then, we can present the competitive ratio of DWM-No Replacement and DWM-Replacement respectively, as we did in the previous section.

We notice that Strategy I, in Model B, works exactly the same way as it does in Model A. Thus, the cost models in both Model A and Model B are identical. This fact leads to

the conclusion that DWM-No Replacement algorithm is still $(1 + C_c + C_d)$-competitive in Heterogenous Object Sizes Model. We omit details of the proof, since the proof for Theorem 1 in previous section still holds for this case.

Each request in a $win(o)$ of an object o is serviced by DWM-Replacement as a non-saving read, a saving read or a write. During the servicing process, when the local database at a processor is full, Strategy II (LRU^{het}) and Strategy III (LFU^{het}) choose more than one object in use instead of only one object to drop in order to make enough room for the newly incoming copy of the object. We assume that each time strategy II and strategy III will evict at most m objects from the local database. Then, we have the following theorem.

Theorem 3 *For any integer $t \geq 1$, DWM-Replacement algorithm is $\left(2 + (2 + m) \cdot \frac{C_c}{C_d}\right)$-competitive.*

In the proof of Theorem 3, we follow the same analysis as we did in the proof of Theorem 2 and prove the following lemma before we realize the result of the theorem. Similarly, we split the proof of the lemma into Part A and Part B according to different types of sub-schedules, say Type I sub-schedule and Type II sub-schedule. For each part of proof, first we establish an upper bound on the communication cost of the DWM-Replacement algorithm. Second, we establish a lower bound on the communication cost of the optimal algorithm for the same schedule. Finally, we take the quotient of these two bounds to derive the competitive ratio of the DWM-Replacement in Model B.

Lemma 3 *Suppose that $IA_o(i)$ is the initial allocation scheme of object o on $\psi_o(i)$ and $\psi_o(i) = \{\psi_o^{ss1}, \psi_o^{ss2}\}$ in $win(o)$. Suppose that $IA_o^*(i)$ is the initial allocation scheme of object o on $\psi_o(i)$ using an optimal algorithm that satisfies t-availability constraint. Then,*

$$COST_{DWM-R}(IA_o(i), \psi_o(i)) \leq \left(2 + (2 + m) \cdot \frac{C_c}{C_d}\right) \cdot COST_{OPT^*}(IA_o^*(i), \psi_o(i))$$

Proof. We split the $win(o)$ into several subschedules, each one of which belongs to Type I sub-schedule ψ_o^{ss1} or Type II sub-schedule ψ_o^{ss2}. Type I sub-schedule consists of read requests only. Type II sub-schedule comprises a write request followed by zero or more read requests. Then, we show that the cost of DWM-R on both types of sub-schedules is at most $\left(2+(2+m)\cdot\frac{C_c}{C_d}\right)$ times as much as the cost of an optimum offline algorithm.

Proof of Part A (for read-only Type I sub-schedule): It follows trivially from the proof of Part A of Lemma 2 in section 4.2.2. Note that each object eviction behavior incurs one control-message cost, say C_c, to inform the CCU of the alteration of the allocation scheme of the object evicted. As we have assumed DWM-Replacement algorithm evicts at most m objects in the processor's local database by either strategy II or strategy III, we just substitute this part of control-message cost with $m \cdot n_p(1) \cdot C_c$ in the proof of Part A of Lemma 2, where $n_p(1)$ is the number of different reading processors of $\psi_o(1)$ which are not in $IA_o(1)$.

Proof of Part B (for Type II sub-schedule): The proof follows trivially from our analysis in Part B of Lemma 2 in section 4.2.2. We also use $m \cdot n_p(i) \cdot C_c$ to calculate the cost of object eviction behaviors when object replacement is needed in the proof process, and replace this part of cost, $n_p(i) \cdot C_c$ in the proof of Part B of Lemma 2, with $m \cdot n_p(i) \cdot C_c$, where $n_p(i)$ is the number of different reading processors of $\psi_o(i)$ which are not in $IA_o(i)$. Then, we obtain

$$COST_{DWM-R}(IA_o(i), \psi_o(i)) \leq \left(2+(2+m)\cdot\frac{C_c}{C_d}\right) \cdot COST_{OPT^*}(IA_o^*(i), \psi_o(i)) + K$$

where K is a constant value.

From both proof of Part A and Part B, Lemma 3 is immediate. □

Proof of Theorem 3: The total cost for servicing the request schedule window $win(o)$ by DWM-Replacement is given by,

$$COST_{DWM-R}(IA_o, \psi_o) = \sum_{i=1}^{n} COST_{DWM-R}(IA_o(i), \psi_o(i)) \tag{4.60}$$

$IA_o(i)$ is the initial allocation scheme of object o on $\psi_o(i)$, which is formed after request schedule $\psi_o(i-1)$ is serviced by DWM-Replacement.

Similarly, the service cost of the same request schedules in $win(o)$ by an optimal offline algorithm is given by,

$$COST_{OPT^*}(IA_o, \psi_o) = \sum_{i=1}^{n} COST_{OPT^*}(IA_o^*(i), \psi_o(i)) \tag{4.61}$$

$IA_o^*(i)$ is the initial allocation scheme of object o on $\psi_o(i)$, which is formed after servicing $\psi_o(i-1)$ using the optimal algorithm. Note that $IA_o(1) = IA_o^*(1) = IA_o$.

Therefore, the proof is evident with the application of Lemma 3.

$$\begin{aligned} COST_{DWM-R}(IA_o, \psi_o) &= \sum_{i=1}^{n} COST_{DWM-R}(IA_o(i), \psi_o(i)) \\ &\leq \left(2 + (2+m) \cdot \frac{C_c}{C_d}\right) \cdot \sum_{i=1}^{n} COST_{OPT^*}(IA_o^*(i), \psi_o(i)) + K \\ &\leq \left(2 + (2+m) \cdot \frac{C_c}{C_d}\right) \cdot COST_{OPT^*}(IA_o^*, \psi_o) + K \end{aligned} \tag{4.62}$$

Hence the proof. □

Chapter 5

Experimental Analysis of the Algorithms

In this section, we will present experimental analysis on the performance of different algorithms in terms of different object replacement policies implemented. We attempt to quantify the performance of different algorithms under these replacement strategies with respect to several influencing parameters such as, local database buffer availability, number of objects in the system, number of servers, and object size, respectively. In previous chapter, we have theoretically analyzed the performances of different strategies by using competitiveness. However, we find that, since the cost functions of DWM-LRU are same as those of DWM-LFU, the competitive ratios of this two algorithms are identical. We have denoted one category by DWM-Replacement in previous chapters to indicate that DWM-LRU and DWM-LFU belong to a same level group if we use competitiveness as a performance analysis tool. Thus, in this chapter, as a supplement, we are eager to evaluate performance of each specific algorithm by simulations. Specifically, we are interested in studying the experimental comparison of the performances of DWM-No Replacement, DWM-LRU and DWM-LFU with each other. The same experimental comparison is also conducted in Model B, for cost comparisons of

DWM-No Replacement, DWM-LRU^{het} and DWM-LFU^{het}.

5.1 Some Basic Views and Expectations on the Experiments

In order to explore the experimental performances of data allocation problem with buffer constraints, we are first eager to know in which aspect the different object replacement policies have impact on the execution of DWM. From our original intention to devise the DWM, the objective of DWM is to service the requests in a cost-effective manner to obtain as low servicing cost as possible. Therefore, DWM services each request with different mechanisms. A read request will be treated as a saving-read request or a non-saving-read request accordingly. However, when local database buffer size at a processor is limited, object replacement strategies will play an important role. Strategy I, namely No Replacement described in Section 3.2, simply refuses to save the new requested object copy into the local database buffer when the buffer has no available space. At first glance, this behavior saves one unit I/O cost (this can be seen from the cost functions in Section 3.3), however, as we introduced above, the DWM decides whether a read request should be treated as a saving-read request or a non-saving-read request according to a judgement on if these saving read requests will save the servicing cost for later requests. Thus, in the long run, Strategy I thoroughly ruins the execution of the DWM and causes an obvious increase of the servicing cost.

Regarding both Strategy II and Strategy III in both Model A and Model B, when local database at a processor has no available space for a new object copy, the strategies will drop some object in use from local database buffer. Unlike Strategy I, Strategy II and Strategy III do not intervene the execution of DWM, which means any saving-read request and write request will result in storing the new copy of the object in the local database as it is required

by DWM. In this case, as the object replacement is inevitable, we hope system retains objects which are more likely to be accessed in near future. From the cost model introduced before, the most expensive cost comes from servicing a remote read request. If we implement a replacement strategy which is to keep the most popular objects residing in the local database, any future access of such objects will be serviced locally. Hence, it saves not only a data transmission cost C_d but also a control-message cost C_c, which is incurred by requesting a copy of the object from remote processors if the future access has to be serviced remotely. Therefore, the ability of an object replacement strategy to pick up and evict an object which has higher probability not to be accessed in the immediate future turns out to be very important.

Although we can not give a clear proof of whether Strategy II performs better than Strategy III or vice versa, we still are able to predict the possible performance of each strategy by exploring their intrinsic natures. The LRU algorithm is a popular policy and often results in a high performance to predict the probability of future access of an object in use. But based on some previous research [36, 37, 39], it is also noticed one of the main weaknesses of LRU, which is the fact that the resident set of local database can be flooded by objects that are referenced only once, flushing out objects with higher probability of being re-accessed. In this sense, LFU has no such worries as LFU tends to keep the objects with more number of references. Nevertheless LFU prevents object, which may not be accessed again, with large reference counts from being dropped. This causes a reduction of the effective size of a local database. Since our simulation is established on a basis of relatively small size of local database capacities compared to the disk buffering and proxy cache used in World Wide Web where LRU and LFU are widely implemented, popular objects have lower probability to get a chance of being accessed over and over again. Thus we are more likely to see that the references to object accessed only once make up large percentile of total references. From this point of view, we expect to see DWM-LRU has a better performance in terms of total servicing cost than DWM-LFU does.

When the same experimental analysis goes to Model B, from our definition and description of the algorithms, LRU$^{\text{het}}$ and LFU$^{\text{het}}$ are the extension of LRU and LFU respectively. Although the size of object is also one consideration, the Strategy II and Strategy III are, in fact, not sensitive to the size of the evicted object candidates. Regarding LRU$^{\text{het}}$, the selected objects are in order of maximum backward distance of most recent reference; regarding LFU$^{\text{het}}$, the picked objects are in order of minimum reference counts. LRU$^{\text{het}}$ and LFU$^{\text{het}}$ actually equally treat each object with different size as the candidates to be evicted. According to this manner, we can expect that all the analysis about LRU and LFU above are also properly used for LRU$^{\text{het}}$ and LFU$^{\text{het}}$. We also expect to see that DWM-LRU$^{\text{het}}$ performs better than DWM-LFU$^{\text{het}}$ in our simulation studies.

To carry out the simulation studies to identify the performance of different algorithms implemented, we choose to write our own simulator, since none of the tools currently available exactly fits our needs. Our simulator simulates multiple distributed databases with local database buffers consisting of fixed-size blocks, at the same time the program also simulates the behavior of different data allocation algorithms under the situations with buffer constrains. The main performance measure used in this study is the cumulative servicing cost of a number of requests. In our simulations, we let $C_{io} = 1$, $C_c = 5$ and $C_d = 10$. We assume that each processor p in the network generates reads and writes of different objects independently of other processors. We yet do not know to what extent the request pattern will exhibit the locality properties which indicate the relationship between recently accessed objects and future requests. Thus, we let simulation run on access patterns that are generated randomly. A random access pattern gives us a relatively chaotic request pattern. First, the requests are serialized by CCU(central control unit), then, as we describe in the chapter 3, request windows $win(o)$ for each specific object o are formed. Each window contains at most one Type I sub-schedule and possibly several Type II sub-schedules. We, then, show our simulation studies in both Model A and Model B in the following sections.

5.2 Simulation Results in Model A

In Model A, we assume all objects are of the same size. Firstly, we set local database storage capacity some fixed value for all processors in distributed system and see what is the result generated by different algorithms implemented with different number of requests. The basic parameters for this first simulation are given in the following Table 5.1.

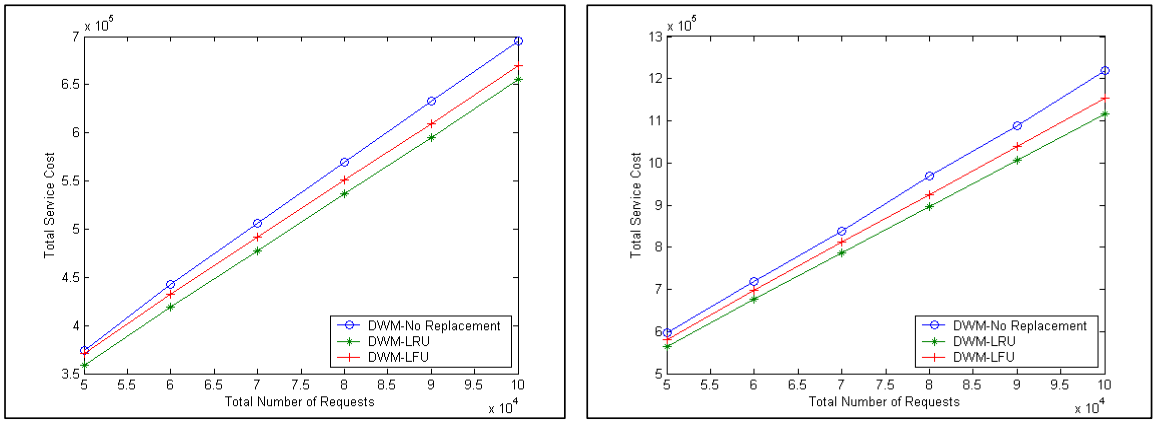
Number of servers	50 (numbered 1 through 50)
Object pool(the number of objects)	200 (numbered 1 through 200)
Object size	1K bytes
Request pattern(the percentage of write requests out of total number of requests)	5% and 10%
Local database capacity(fixed)	50K bytes
Number of requests(a range)	50,000 - 100,000

Table 5.1: Parameters for simulation given a fixed local database capacity

We assume that distributed database system contains 50 processors. The local database size for each processor is set to 50K bytes. The object pool includes 200 objects and each object size is set to 1k bytes. Each of processors issue requests for the object from the object pool randomly. The total number of requests ranges from 50,000 to 100,000. The read-write pattern is described by the percentage of write requests out of total number of requests. In our simulation, we assume it is a read-intensive network. Regarding a write-intensive network, since DWM invalidates obsolete copies of the object in different processors when it deals with a write request according to its own mechanism, the large number of write requests will produce lots of free space in local databases at processors. Grounded on this assumption, let the simulation run under situations where write requests are 5% and 10% of total number of requests respectively. Fig. 5.1 clearly shows the simulation results.

For readers who feel that local database storage capacity, as well as the object pool with

200 objects with each size of 1K bytes and number of requests from 50,000 to 100,000 are unrealistically small for modern applications, note that same results hold if all these numbers are multiplied by a constant. The smaller numbers were used in simulation to save effort.



Cumulative cost under different number of requests (write requests: 5%)

Cumulative cost under different number of requests (write requests: 10%)

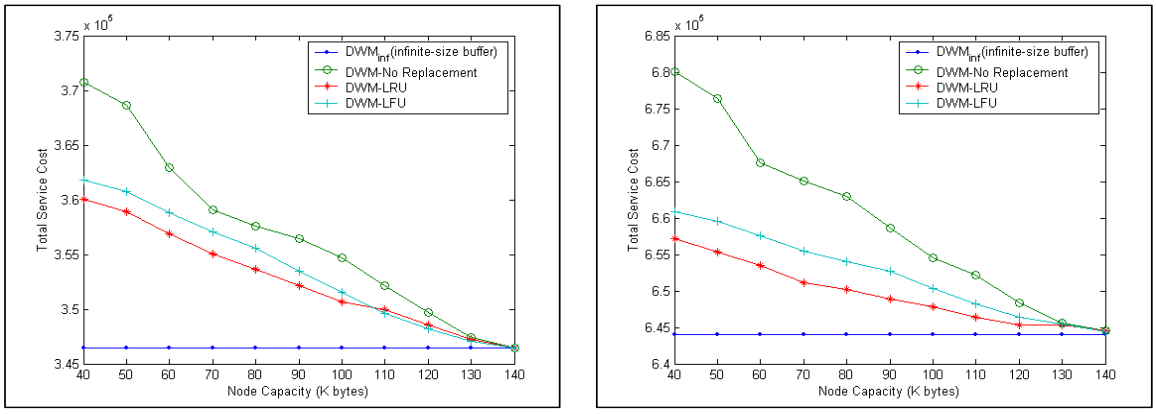
Figure 5.1: Cumulative cost under different total number of requests (write requests: 5% and 10%)

In addition, we also want to compare the performances of different algorithms under different node capacities when we set the total number of requests fixed. In this simulation comparison, we assume a base case without local database buffer constraints, denoted by $\mathrm{DWM}_{\mathrm{inf}}$, which demonstrates the servicing cost of DWM based on the assumptions that processors' local database size is infinite. The parameters of the simulation are described in Table 5.2. The results are exhibited in Fig. 5.2.

After comparing the results, we can make the following observations. It can be seen that, from Fig. 5.1, with the increase of number of total requests, the cost of serving requests by DWM-No Replacement, DWM-LRU and DW-LFU increases monotonously. This is easy to understand. Because when the total number of requests increases, no matter which algorithm is implemented, the total number of requests handled by distributed database system increases and any single one of services invokes certain cost. Thus, the cumulative cost is a monotonous

Number of servers	50 (numbered 1 through 50)
Object pool(the number of objects)	200 (numbered 1 through 200)
Object size	1K bytes
Request pattern(the percentage of write requests out of total number of requests)	5%
Number of requests(fixed)	50,000 and 100,000
Local database storage capacity(a range)	40K - 140K bytes

Table 5.2: Parameters for simulation given a fixed number of requests



Cumulative cost under different database capacities (number of requests:50k)

Cumulative cost under different database capacities (number of requests:100k)

Figure 5.2: Cumulative cost under different local database capacities (fixed total number of requests of 50k and 100k respectively and write requests: 5%)

accretion process with the increase of number of requests. We have found that the rate of increase of cumulative servicing cost of DWM-No Replacement is much higher than the other two algorithms. This is because, given a fixed local database capacity, the larger the total number of requests arriving to the system, the more serious the execution of decisions made by DWM is intervened, since once the local database of a processor has no available space, any later request will never result in saving any object in local database at the processor. As the figure shows, DWM-No Replacement performs worst in all cases and DWM-LRU algorithm consistently outperforms the corresponding DWM-LFU. This tendency is clear when the write

requests is made up of 5% of total number of request, while LRU is tenderly better than the corresponding LFU with DWM when write requests is 10% of total number of requests. These findings confirm our expectations in our previous discussion which is LRU may perform better than LFU as an object replacement strategy in our given simulation environment.

In Fig. 5.2, it is clearly shown that DWM-No Replacement has the worst performance and DWM-LRU performs better than DWM-LFU, especially when the local database buffer size is relatively small. The cost of servicing requests by DWM without considering any local database capacity constraints acts as a lower bound for performance of DWM-No Replacement, DWM-LRU and DWM-LFU. We can see from the figure, given a fixed number of requests, as the local database capacity becomes larger, the servicing cost of DWM-No Replacement, DWM-LRU and DWM-LFU will be closer to the servicing cost of DWM with infinite buffer capacity. Finally, when the database buffer capacity is large enough, we can see the convergence of the costs by different algorithms. From this analysis, we can have the conclusion that the replacement algorithms exert bigger effects under small database storage capacities, because more object replacements happen when the local database buffer size is small. With the increase of the total number of requests, this tendency becomes much stronger.

We find that the DWM-LFU outperforms the DWM-LRU slightly when local database capacity goes over 110K bytes in Fig. 5.2. This is possibly because that when node capacity becomes larger, popular objects have a better chance to stay in the system for a long time to get a reference again, thus increase their likelihood of staying in the server due to the characteristic of LFU. However, after the node capacity is larger than 110K bytes, the cost difference between DWM-LFU and DWM-LRU is not very obvious as the cost has started to get close to the cost bound by $\mathrm{DWM}_{\mathrm{inf}}$ without node capacity constraints.

It is also interesting to find, from Fig. 5.2, that we are able to know the smallest local database buffer capacity needed by sole implementation of DWM under the parameters we conduct our

simulations. For example, in the left figure in Fig. 5.2, when the local database buffer capacity increase to 140K bytes, there is no cost difference between DWM_{inf}, DWM-No Replacement, DWM-LRU and DWM-LFU, which means the current node capacity is enough for DWM to execute its mechanism normally without the need to consider the buffer constraints. It possibly means that the size of 140K bytes is the lowest requirement to run DWM solely without any aid of object replacement strategies with the given number of servers, number of objects, object size and read-write request pattern.

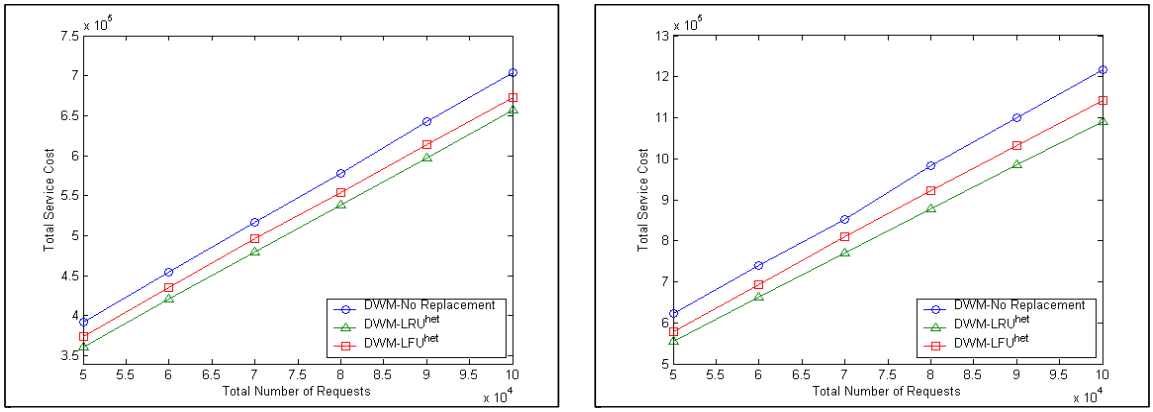
5.3 Simulation Results in Model B

In Model B, we assume that all objects are of different sizes. When considering that the objects have different sizes, here we define the object size in a range of 1K bytes to 5K bytes. The object pool is the same as we described before. Each object size is uniformly distributed in the object size range. Same as we analysis the performances of different algorithms in Section 5.2, here we also divide our simulation into two parts. Firstly, we fix the local database buffer capacity and see the simulation results under different total number of requests. The primary parameters is shown in Table 5.3. It is noted that the fixed node capacity becomes larger just because the object size changes in a value range between 1K and 5K bytes. The simulation results are shown in the Fig. 5.3.

Then, we observe the simulation results with fixed number of requests and varied local database buffer capacities. Table 5.4 contains the parameters under which the simulation is conducted. Fig. 5.4 depicts the comparison of cumulative servicing cost by different algorithms under these situations. As we perform the analysis in Section 5.2, we also assume a base case of the comparisons, denote by DWM_{inf}, to indicate the servicing cost of DWM under the assumption that local database buffer size is infinite.

Number of servers	50 (numbered 1 through 50)
Object pool(the number of objects)	200 (numbered 1 through 200)
Object size (uniform distribution)	1K - 5K bytes
Request pattern (the percentage of write requests out of total number of requests)	5% and 10%
Local database capacity (fixed)	120K bytes
Number of requests(a range)	50,000 - 100,000

Table 5.3: Parameters for simulation for heterogenous-sized objects given a fixed local database capacity



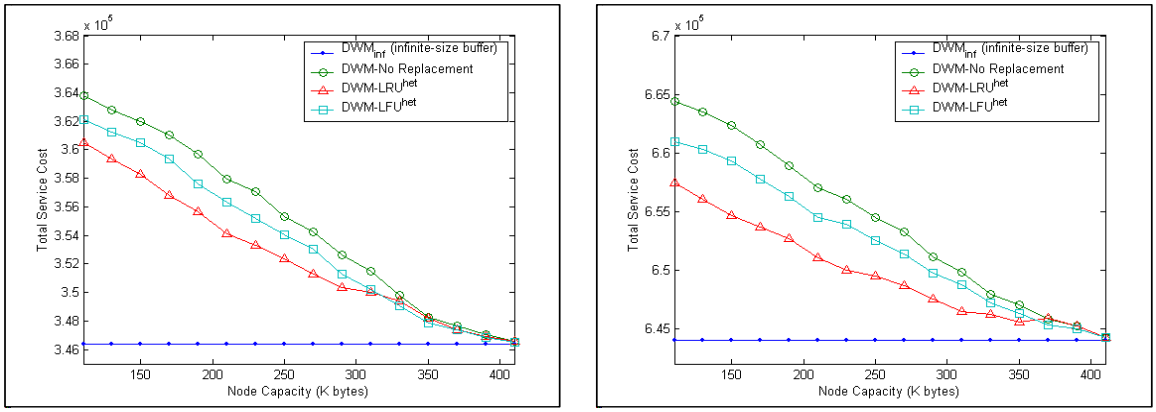
Cumulative cost under different number of requests (write requests: 5%)

Cumulative cost under different number of requests (write requests: 10%)

Figure 5.3: Cumulative cost under different total number of requests in Model B (write requests: 5% and 10%)

Number of servers	50 (numbered 1 through 50)
Object pool(the number of objects)	200 (numbered 1 through 200)
Object size (uniform distribution)	1K - 5K bytes
Request pattern(the percentage of write requests out of total number of requests)	5%
Number of requests(fixed)	50,000 and 100,000
Local database storage capacity (a range)	110K - 410K bytes

Table 5.4: Parameters for simulation for heterogenous-sized objects given a fixed number of requests



Cumulative cost under different database capacities (number of requests:50k)

Cumulative cost under different database capacities (number of requests:100k)

Figure 5.4: Cumulative cost under different local database capacities in Model B (fixed total number of requests of 50k and 100k respectively and write request: 5%)

From the figures presented in this section, we find that DWM-LRU^{het} gains relatively lower servicing cost than DWM-LFU^{het} in this simulation model. The DWM-No Replacement still obtains the worst performance as its servicing cost under any situations is the highest. We also observe that the curves' tendency is quite similar as the curves in figures in Section 5.2. This illustrates that both LRU^{het} and LFU^{het} have no special sensibility to the object size, which means object size is not a key factor to affect the behavior of LRU^{het} and LFU^{het} when they are looking for proper candidates to evict. This has been mentioned in Section 5.1 and we have expected the results as this four figures illustrate. An interesting phenomenon is that we find that the performance improvement by DWM-LRU and DWM-LFU compared with DWM-No Replacement is not as quite distinct as the bigger improvement shown in simulation results in Model A. We conjecture the reason is that LRU^{het} and LFU^{het} may need to invalidate more than one object in the local database at a processor to guarantee the enough space for the new object replica. However, in Model A, as all object sizes are identical, when processors' local database is full, only one object need to be evicted. From our description of the cost functions in Section 3.3, every eviction of the object will incur one additional control-message cost. But this situation has no ill effect on DWM-No Replacement algorithm, as it does not evict any object in use at all. Thus, we can see the cost difference between DWM-LRU, DWM-LFU and DWM-No Replacement is smaller than that shown in figures in Section 5.2.

Chapter 6

Conclusions and Future Work

In this thesis, we have addressed important issues of distributed data management (DOM) in the domain of distributed database system (DDBS). The purpose of adaptive data allocation and replication algorithms is to improve the performance of distributed database systems by adjusting the allocation scheme for an object (i.e. the number of replicas of the object, and their locations) to the current access pattern in the network. With the objective of minimize the total servicing cost of the incoming requests, we consider the problem of dynamic data allocation and replication in a distributed database system with local database buffer constraints.

For servicing requests that arrive at a distributed database system, we proposed a practical algorithm, DWM, for dynamic data allocation and replication, which adapts the allocation scheme of the object to varied read-write request patterns. We described how it is implemented in a distributed fashion. The working style of this algorithm uses a systematic procedure to extract and organize the requests in multiple request windows and processing each of the request windows concurrently. DWM basically alters the allocation scheme, as per the request pattern. This means that the processors are added to or removed from the allocation scheme of an object at each time the requests in the sub-schedule of this object are serviced. The central

idea of DWM algorithm is to decide a read request as a saving-read request or not depending on whether it can minimize the cost of servicing all the requests in a sub-schedule. Further, as we take local database storage limitations at the various processors into consideration, three strategies are proposed to cope with the situation when local database buffer has no available space for the new storage. Strategy I, No Replacement, simply denies any object replacement. The new copy of the object that is ordered by DWM to be replicated will not be saved in the local database at the processor if this processor does not have enough storage space. On the contrary, Strategy II and III basically drop some object in the local database buffer at the processor when space is needed according to the order for this processor to store a new copy of the object. Specifically, LRU selects the best candidates to evict mainly by exploiting the principle of temporal locality and drops the object used least in the recent past. LFU uses the history of accesses to predict the probability of a later re-access of an object and drops the object used least frequently. Both of the above strategies are used in the scenario in which all objects are of same sizes (Homogeneous object sizes Model). We also attack the problem in the situation in which different objects have different sizes (Heterogeneous object sizes Model). In the later model, we design two replacement algorithms based on principles of LRU and LFU respectively, namely LRU^{het} and LFU^{het}.

We analyzed (theoretically and experimentally) the performance of different algorithms with local database buffer constraints. In our theoretical analysis, we use competitiveness, which is widely used in the analysis of online algorithm, to evaluate the performance of different algorithms, say DWM-No Replacement and DWM-Replacement. Note that DWM-Replacement is defined to group DWM-LRU and DWM-LFU as both of them share the same cost functions shown in Section 3.3. We showed that in the model where communication and I/O costs are considered, the DWM-No Replacement is $(1 + C_c + C_d)$-competitive, the DWM-Replacement is $(2 + 3 \cdot \frac{C_c}{C_d})$-competitive. C_c is the ratio of the cost of transmitting a control message to the cost of inputting/outputting the object to the local database and C_d is the ratio of the

cost of transmitting the object between two processors to the I/O cost. A DOM algorithm is c-competitive if the ratio of the cost of the algorithm to the cost of an optimal offline algorithm is at most c for an arbitrary sequence of read-write requests. The results of the comparison between the DWM-No Replacement and DWM-Replacment algorithms are that DWM-No Replacement is superior when $(C_c + C_d) < 1$, and DWM-Replacement is superior when $(C_c + C_d) > 4$. These results are summarized in Fig. 4.2. The area in Fig. 4.2 that is marked "unknown" represents the C_c and C_d values for which it is currently unknown whether the DWM-Replacement algorithm is superior to the DWM-No Replacement algorithm or vice versa. The reason for this uncertainty is that the radio of C_c/C_d is uncertain and sensitive for the comparison.

We conducted extensive simulation experiments and used cumulative cost to quantify the overall performance of different algorithms. Our experimental results showed the following. For a randomly generated read extensive request pattern and assumptive value of C_{io}, C_c and C_d, in both Model A and Model B, DWM-No Replacement performs the worst with the highest servicing cost. This observation gives us a clear hint that a kind of object replacement mechanism is quite necessary when we consider the data allocation problem with buffer constraints. This is because No Replacement interrupts the execution of the decisions made by DWM eventually. We also find that DWM-LRU performs clearly better than DWM-LFU in our simulation. This result is expected from our analysis in Section 5.1. The same simulation results were obtained from the simulation in the heterogeneous-object-sized environment. These results act in accordance with our analysis as LRU^{het} and LFU^{het} inherit the principles of LRU and LFU. Although we have known that, in environment Model B, making room for a new object replica may cause more than one objects in use at a processor to be evicted due to the different sizes of different objects and directly lead to the extra control-message cost, if the number of requests in simulation is large enough, this part of extra cost of DWM-LRU^{het} is nearly identical to that of DWM-LFU^{het} . Therefore, if LRU^{het} shows a higher ability to keep

the more popular objects in the local database at the processors than LFU^{het}, the servicing cost of DWM-LRU^{het} is lower than that of DWM-LFU^{het}. However, it must be noticed that the efficiency of a replacement algorithm depends on the object trace or run-time behavior of programs, we could not have a fixed superiority of one replacement algorithm applied with DWM over the other because of this kind of dependence.

The research described in this thesis serves as a starting point for a rigorous model for the data allocation and replication scheme with local database storage constraints. We find several directions need further work. One of them is to introduce more precise request access patterns which can precisely represent the data access pattern in the real distributed database system. We also want to implement some complicated replacement algorithms to fit well with different request access patterns to help allocation and replication algorithms work at as low cost as possible. Especially, when considering the Heterogeneous object sizes Model, a replacement algorithm which is sensitive to the size of objects is likely to be more efficient in our expectation. Another important work that remains to be addressed is to explore whether or not a single relationship could be derived to link data replication with database storage capacity constraints, if possible, also with all other parameters such as network bandwidth and different object sizes, respectively. Such an explicit relationship, without demanding the application of data allocation algorithm and object replacement strategies independently, is expected to be more efficient to deliver a best performance.

Bibliography

[1] M.Stonebraker, "Readings in Database Systems," San Mateo, Calif.:Morgan Kaufman, 1988.

[2] M.Tamer Özsu and Patrick Valduriez, "Principles of Distributed Database Systems," Upper Saddle River, NJ : Prentice Hall, 1999.

[3] P.S.Yu, M.S.Chen, and D.D.Kandlur, "Grouped Sweeping Scheduling for DASD-based Multimedia Storage Management," *Multimedie system*, 1(3),99-109,1993

[4] D.L.Black and D.D.Sleator, "Competitive Algorithms for Replication and Migration Problems," *Technical Report CMU-CS-89-201*, Carnegie Mellon Univ., Nov.1989.

[5] R.L. Graham, "Bounds for Certain Multiprocessor Anomalies," *Bell System Technical Journal*, 45:1563-1581, 1996.

[6] M.Manasse, L.A. McGeoch, and D. Sleator, "Competitive Algorithms for Online Problems," *Proc. 20th ACM Symp. Theory of Computing*, pp.322-333, 1988.

[7] A.Fiat, R.Karp, M.Luby, L.A.McGeoch, D.Sleator, and N.E.Yong, "Competitive Paging Algorithms," *J.Algorithms*, vol.12, pp.685-699, 1991.

[8] Allan Borodin and Ran El-Yaniv, "Online Computation and Competitive Analysis," New York: Cambridge University Press, 1998.

[9] P.Verissimo and L.Rodrigues, "Distributed Systems for System Architects," *Kluwer Academic Publishers*, USA, 2001.

[10] Andrew S.Tanenbaum and Maarten Van Steen, "Distributed Systems: Principles and Paradigms," *Prentice Hall*, 2002.

[11] O.Wolfson and Y.Huang, "Competitive Analysis of Caching in Distributed Databases," *IEEE Transactions on Parallel and Distributed Systems*, Vol.9, No.4, pp.391-409, April 1998.

[12] O.Wolfson, Shushil Ja jodia and Yixiu Huang, "An Adaptive Data Replication Algorithm", *ACM Transactions on Database Systems*, 22(2):255-314, 1997.

[13] O.Wolfson, Shushil Jajodia, "Distributed Algorithms for Dynamic Replication of Data," *Proc. ACM-PODS*, 1992.

[14] Y.Huang, P.Sistla, and O.Wolfson, "Data Replication for Mobile Computers," *Proc. ACM-SIGMOD Int'l Conf. Management of Data*, Minneapolis, Minn., May 1994.

[15] Edward G. Coffman, Jr. and Peter J.Denning, "Operating Systems Theory," *Prentice Hall*, 1973

[16] Kai Hwang, "Advanced Computer Architecture: Parallelism, Scalability, Programmability," *McGraw-Hill*, 1993

[17] Y.Huang, and O.Wolfson, "A Competitive Dynamic Data Replication Algorithm," *IEEE Proc. Ninth Int'l Conf. Data Eng.'93*, pp.310-317, Vienna,1993.

[18] Baruch Awerbuch, Yair Bartal, and Amos Fiat, "Competitive Distributed File Allocation", In *Proc. 25th ACM Symp. on Theory of Computing*, pages 164-173, May 1993.

[19] Swarup Acharya and Stanley B.Zdonik, "An Efficient Scheme for Dynamic Data Replication", *Technical Report CS-93-43*, Department of Computer Science, Brown Universtiy, September 1993

[20] T.M.Ozsu and P.Valduriez, "Principles of Distributed Database Systems," *Patrick Valduriez Prentice Hall*, 1999.

[21] S.Albers, "Competitive Online Algorithms," *BRICS LS-96-2, LSSN 1295-2048*, September 1996.

[22] Won.Y.J and Srivastava.J, "Strategic Replication of Video Files in a Distributed Environment," *Multimedia Tools and Applications*, (2)pp.249-283 March 1999.

[23] P.Bernstein, V.Hadzilacos, and N.Goodman, "Concurrency Control and Recovery in Database Systems," *Addison-Wesley*, 1987.

[24] Naser S.Barghouti and Gail E.Kaiser, "Concurrency Control in Advanced Database Applications," *ACM Computing Serveys*, 23(3):269-317, September 1991.

[25] W.Cellary, E.Gelenbe and T.Morzy, "Concurrency control in distributed database systems," Amsterdam; New York: North-Holland; New York: Sole distributors for the U.S.A. and Canada, Elsevier Science Pub. Co., 1988.

[26] Haritsa,J.R., Carey,M. and Livny,M., "Data Access Scheduling in Firm Real-Time Database Systems". *The Journal of Real-Time Systems*, 4, pp.203-241, 1992.

[27] M.J.Franklin, "Client Data Caching," *Kluwer Academic Publishers*, 1996.

[28] M.Zahariousdakis and M.J.Carey, "Hierarchical, Adaptive Cache Consistency in a Page Server OODBMS," *Proc. Int'l Conf. Distributed Computing System*, 1997.

[29] B.Veeravalli, "Document Caching Policies on High-Speed Distributed Networks for Personalized Multimedia Services". *IEEE International Conference on Networks, ICON 2000*, pp.215-219, Singapore, 2000.

[30] C.Liu and P.Cao, "Maintaining Strong Cache Consistency in the World-Wide Web", *Proc. Int'l Conf. Distributed Computing Systems*, 1997

[31] A.Chankhunthod, P.B.Danzig, C.Neerdaels, M.F.Schwarts, and K.j.Worrell, "A Hierarchical Internet Object Cache," *Proceedings of the 1996 USENIX Technical Conference*, Jan. 1996

[32] C.Aggarwal, J.L.Wolf and Yu, P.S., "Caching on the world Wide Web," *IEEE Transactions on Knowledge and Data Engineering*, Vol.11 1, Jan.-Feb. 1999, Page(s):94-107

[33] A.belloum and L.O.Hertzberger, "Replacement Strategies in Web Caching," *Conference ISIC/CIRA/ISAS'98*, Gaithersburg, September 1998, Maryland USA.

[34] S.Jin and A.Bestavros, "Popularity-Aware GreedyDual-Size Web Caching Algorithms," *Technical Report TR-99/09*, Computer Science Department, Boston University, 1999.

[35] R.karedla, J.S.Love, and B.G.Wheery, "Caching Strategies to Improve Disk System Performance," *IEEE Computer*, 27(3):38-46, March 1994.

[36] E.J.O'Neil, P.E.O'Neil, and G.Weikum, "The LRU-K Page Replacement Algorithm for Database Disk Buffering," In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 297-306, New York, 1993.

[37] L.Rizzo and L.Vicisano, "Replacement Policies for a Proxy Cache," *Technical report, rn/98/13*, University College London, Department of Computer Science, Gower Street, London WC1E 6BT, UK, 1998. http://www.iet.unipi.it/luigi/caching.ps.gz.

[38] J.T.Robinson and M.V. Devarakonda, "Data Cache Management Using Frequency-based Replacement". *Performance Evaluation Review*, 18(1):134-142,May 1990.

[39] Alfred V. Aho, Peter J. Denning, and Jeffrey D. Ullman, "Principles of Optimal Page Replacement," *J. ACM*, v. 18, no. 1, pp.80-93, 1971.

[40] O.Wolfson and A.Milo, "The multicast Policy and Its Relationship to Replicated Data Placement," *ACM TODS*, 16(1), 1991.

[41] L.W.Dowdy and D.V.Foster, "Comparative Models of the File Assignment Problem," *ACM Computing Surveys*, 14(2), 1982.

[42] D.Sleator and R.Tarjan, "Amortized Efficiency of List Update and Paging Rules," *CACM* 28(2), 1985.

[43] A.Karlin, M.Manasse, L.Rudolph, D.Sleator, "Competitive Snoopy Caching," *Algorithmica*, 1998.

Appendix A: Author's Papers

[1] Gu Xin, Bharadwaj Veeravalli, and Lin Wujuan "Design and Analysis of Practically Realizable Dynamic Data Allocation and Replication Algorithm with Buffer Constraints for Distributed Databases", submitted to *IEEE Transactions on Parallel and Distributed Systems.*

[2] Gu Xin, and Bharadwaj Veeravalli, "An Efficient Data Allocation and Replication Algorithm Considering Dynamic Primary Servicing Server Set", submitted to *Cluster Computing.*